BASSEM HAMDI

Développement d'une grille hexagonale hiérarchique et d'algorithmes de clustering « géosémantique » pour l'analyse et la découverte de connaissances géo-spatiales.

Mémoire présentée
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en informatique
pour l'obtention du grade de Maître ès Science (M.Sc)

DÉPARTEMENT D'INFORMATIQUE ET DE GÉNIE LOGICIEL
FACULTÉ DE SCIENCE ET GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

2009

Résumé

Dans le cadre du projet *MUSCAMAGS*[1] *« Multi-scale multi-agent geo-simulation »*, les simulations sont produites dans un environnement virtuel géographique (EVG) qui reflète la réalité géographique grâce à l'usage de données géoréférencées. Compte tenu des applications de mobilité urbaine visées dans ce projet et de la disponibilité des données, l'EVG a été représenté par une grille hexagonale. Cependant, bien qu'il réduise le biais directionnel lors de l'analyse spatiale, ce genre de grille présente un inconvénient important: il ne permet pas une représentation multi-échelle de l'environnement géographique. Dans le cadre de ce projet de maîtrise, nous proposons une autre solution à ce problème. En effet, nous proposons de partitionner l'environnement à l'aide de cellules dont la forme géométrique fondamentale est le triangle équilatéral. Ensuite, à partir de ces cellules, nous développons un algorithme pour créer des cellules hexagonales hiérarchiques selon un indexage conforme à l'approche *column-ordering*. Ensuite nous intégrons ces grilles dans une application de système d'information géographique que nous enrichisson par des techniques d'intelligence artificielle afin de faciliter la découverte et l'interprétation des phénomènes urbains. En effet, nous avons considéré plus particulièrement les automates cellulaires et les techniques de clustering issues du data mining. Ainsi, nous avons exploré une technique de regroupement « géo-sémantique» des cellules en nous basant sur un algorithme de clustering par fusion. Également, nous avons associé aux grilles hexagonales hiérarchiques des automates cellulaires afin d'obtenir un processus de regroupement automatique (auto-regroupement) qui puisse être utilisé pour l'analyse des données spatiales.

[1] http://www2.ift.ulaval.ca/muscamags/

Abstract

In the *MUSCAMAGS*[2] *« Multi-scale multi-agent geo-simulation »* project, simulations are produced in a virtual geographic environment (VGE) which reflects the geographical reality thanks to the use of georeferenced data. Considering urban mobility applications and available data, this EVG was represented by a hexagonal grid. However, although it reduces the directional bias when carrying out spatial analyses, this kind of grid presents a major drawback: it does not allow a multi-scale representation of the geographical environment. In this Master's project, we propose a solution to this problem. Indeed, we propose to create the environment with cells whose geometry is a simple equilateral triangle. Then, using such cells, we developed an algorithm to create hexagonal hierarchical cells according to the column-ordering indexing approach.

Morover, we were able to integrate such grids in a GIS application that we enhanced with artificial intelligence techniques in order to help the discovery and interpretation of urban phenomena. Indeed, we considered particularly cellular automata and clustering data mining techniques. Thus, we explored a geo-semantic clustering technique of the cells, based on a fusion clustering. Finally, we associated cellular automata with hexagonal grids, in order to obtain an automatic clustering process that can be used for spatial data analysis.

[2] [2] http://www2.ift.ulaval.ca/muscamags/

Je dédie ce mémoire à ma mère Soufia,
À mon père Abd-El-Karim,
À ma sœur Asma et mon frère Ghassen,
À ma future douce moitiée.

Table des matières

Liste des tableaux

Liste des figures

Chapitre 1 Introduction

Dans ce chapitre nous présentons le contexte général de notre recherche ainsi que les problèmes spécifiques à notre problématique. Ensuite, nous présentons les objectifs de notre projet et finalement le contenu de ce mémoire.

1.1 Mise en contexte

De nos jours, les intervenants et les décideurs de plusieurs secteurs d'activités (télécommunication, militaire, industrie, médicine) doivent gérer des situations impliquant un grand nombre de personnes dans un espace géographique ciblé. Certaines de ces situations sont en lien avec la sécurité publique (inondation, tremblement de terre), d'autres avec l'ordre public (manifestation, évacuation de bâtiments) ou avec l'utilisation des infrastructures urbaines (gestion du trafic, planification urbaine). La complexité de ce processus demande des méthodes et des approches d'avant garde pour simuler les différents scénarios possibles de dynamique urbaine et afin de fournir un modèle fiable et réaliste pour aider les décideurs à prendre des décisions averties. À cet effet, on utilise généralement les plateformes de géo-simulation. La géo-simulation représente un domaine de recherche qui s'intéresse à la construction d'applications (souvent orientées-objets) qui servent à explorer des idées et des hypothèses dans le but de résoudre divers types de problèmes géographiques (Benenson et al 2005). Cette approche tire profit des avancées en informatique et en géomatique en fournissant des modèles qui tiennent compte à la fois du temps et de l'espace. Par ailleurs, elle combine les concepts et les techniques de l'intelligence artificielle et celles de la géomatique afin de créer des systèmes permettant de simuler des situations complexes (ex : simulation de la foule ou du trafic routier en milieu urbain, propagation des feux de forêts, ...) à des échelles spatiales et temporelles variées. Ainsi, cette approche permet la modélisation d'un système dont les objets élémentaires évoluent au cours du temps, comme par exemple les individus (automates ou agents) ou les entités de l'environnement tels que les bâtiments et les routes. L'objectif de ce type de système est d'étudier les effets de comportements individuels (niveau « micro ») sur des phénomènes observables à de plus hauts niveaux d'organisation géographique (niveau « macro »).

Un autre paradigme utilisé en géo-simulation est celui des automates cellulaires (Wolfram, 1984). Ces derniers permettent de simuler la propagation des phénomènes urbains par l'entremise de leurs états. Par exemple, les automates cellulaires simulent généralement l'évolution de l'occupation des sols et sont utilisés pour comparer divers scénarios d'aménagement urbain. En effet, ces automates ont des états et des caractéristiques qui évoluent dans le temps en fonction de règles pré-établies associées aux cellules de l'automate. À cet égard, la l'abstraction de l'environnement géographique, qui est aussi appelée tessellation spatiale ou grille est importante. Cette dernière est divisée en cellules de formes régulière ou irrégulière. La grille se définit comme une surface qui correspond au recouvrement de d'une portion d'espace à l'aide d'un arrangement de polygones qui ne se chevauchent pas (Worboys, 2004). L'expression « grille régulière » fait référence à une structure dont les polygones sont réguliers, c'est-à-dire tous les côtés et les angles de ces polygones sont égaux (Bell, 1983). Dans le cas contraire, il s'agit d'une grille dite « irrégulière » (Bell, 1983). Cependant, La grande majorité des plateformes de simulation emploient des grilles régulières. Ainsi, l'environnement de simulation est subdivisé en une structure qui est formée par des cellules identiques et un mécanisme d'indexation est utilisé afin d'identifier ces cellules. La grande majorité du temps, ces grilles ont des cellules dont la forme géométrique est le carré.

1.2 Problématiques :

Premièrement, outre la grille à mailles carrées, dans certains cas, on utilise la grille hexagonale pour également abstraire l'environnement géographique, comme celle qui est présentée par la figure 1.1. Toutefois, cette grille n'est pas hiérarchique, c'est-à-dire que les grilles hexagonales ne peuvent pas se décomposer en d'autres sous-cellules de forme hexagonales et de tailles inférieures (Bell, 1983). Dans la littérature, la notion de grille hiérarchique est abordée par plusieurs auteurs comme Paris (Paris, 2007), Samet (Samet,1999) ou Tsui (Tsui, 2002) qui ont utilisé des grilles carrés hiérarchiques. Ceci a comme avantage de pouvoir transiter d'une échelle à une autre au sein de la même grille sans utiliser d'autres structures pour chaque niveau de résolution et par conséquent avoir une solution plus flexible et robuste. Cependant, les grilles à mailles carrés introduisent un

biais directionnel (dans le sens des axes de la grille) pour l'analyse des phénomènes spatiaux. Pour remédier à ce problème de biais, on utilise les grilles hexagonales comme par exemple dans le projet *MUSCAMAGS*. Ce dernier est un projet de recherche qui vise à fournir aux décideurs des outils logiciels de géo-simulation afin de supporter leur processus de prise de décision et ce, dans plusieurs domaines tels que le domaine militaire, médical, industriel, ou de transport. Une grille hexagonale permet de minimiser le biais directionnel à cause de ses propriétés d'adjacence (l'hexagone se rapproche du cercle et comporte des voisins dans six directions). Ces propriétés simplifient les calculs statistiques au niveau des cellules.

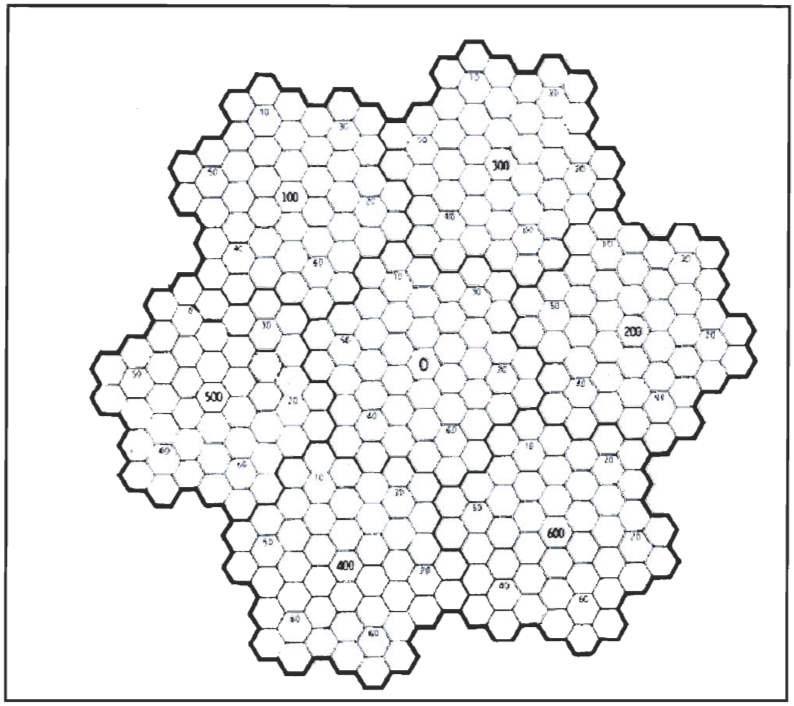


Figure 1.1 **Un exemple d'une grille hexagonale (Gibson, 1982)**

Finalement, la plupart des automates cellulaires sont associés à des grilles régulières sans pour autant prendre en considération le facteur multi-échelles. Par exemple, on utilise un des modèles uni-résolution, (c'est-à-dire qu'il tient compte d'un seul niveau à la fois) pour prédire l'occupation des sols avec les automates cellulaires (Benenson et al, 2005). Par multi-échelles on sous-entend que les cellules peuvent avoir plusieurs niveaux d'abstraction et par conséquent elles créent ainsi une hiérarchie. En effet, chaque cellule

d'un niveau supérieur est divisée en des sous-cellules qui appartiennent à un niveau inférieur et dont la taille est plus petite que celle de la cellule de plus haut niveau. C'est au carrefour du projet *MUSCAMAGS,* des grilles multi-résolution ou hiérarchiques et des automates cellulaires que se situe le présent mémoire. En effet, nous chercherons à intégrer ces trois paradigmes en un seul framework qui puisse être utilisé par un logiciel de simulation ou par un logiciel d'analyse de données. La section suivante décrit plus précisément les objectifs de ce projet.

1.3 Objectifs de la recherche :

Dans le cadre du projet MUSCAMAGS, un projet de doctorat (Chaker, 2009) développe une approche et des outils pour générer des données qui représentent une population synthétique de la ville de Québec. Dans ce projet, on simule les déplacements des personnes de la ville de Québec. Par ailleurs, les lieux de résidences des ménages de cette population sont localisés sur une grille hexagonale qui couvre tout l'espace géographique de la ville de Québec. Notre projet est une recherche exploratoire visant à déterminer comment tirer parti d'une grille hexagonale hiérarchique pour l'analyse à différentes échelles des données reliées à des applications de dynamique urbaine comme celles traitées dans le projet *MUSCAMAGS*. Dans cette optique, nous tenterons de :

- Palier aux limites des grilles hexagonales qui sont utilisées dans le projet *MUSCAMAGS* en travaillant sur des grilles dont l'unité géométrique de base est un triangle équilatéral.
- Avoir un mécanisme de construction générique de ce genre de grilles, c'est-à-dire pouvoir paramétrer ces grilles selon l'usage et le contexte de l'application.
- Préparer un jeu de données propice à l'étude d'un phénomène urbain d'intérêt et ce, à partir des données de la population synthétique du projet *MUSCAMAGS*.
- Assigner des automates cellulaires aux grilles hiérarchiques dans le but de créer un mécanisme de regroupement intelligent pour des fins d'exploration des connaissances, l'analyse spatiale des données et de visualisation de phénomènes émergeants.

- Avoir une interface usager qui répond bien au contexte d'utilisation et qui fournit un mécanisme de visualisation naturel de la hiérarchie à l'aide de la technique de « pan/zoom ».

1.4 Démarche suivie

La méthodologie proposée pour cette recherche s'inspire des méthodes de développement logiciel déjà éprouvées pour la conception de systèmes d'information (Strohmeier,1996) puisque la présente recherche peut être assimilée à la version réduite d'un tel projet. Nous nous proposons donc d'employer une méthode de développement agile et par prototypage (voir Tableau 1.1).

Phase	Étape	Détails
Conception	Analyse des besoins et faisabilité	Revue de littérature et recherche de solutions aux problèmes posés
	Spécifications	Choix des différents paramètres (formes, voisinages ,patron etc.)
	Conception de l'architecture	Conception du modèle UML
Cycles de Réalisation	Conception détaillée	Conception des algorithmes
	Implémentation	Programmation du framework
Validation	Essais	Validation du prototype

Tableau 1-1 Principales étapes de développement proposées

Une approche plus théorique sera adoptée à l'amorce de la recherche afin de bien cerner le contexte des applications de géo-simulation et, plus spécifiquement, les diverses fonctions attribuées aux grilles. Cette recherche sera réalisée en se concentrant sur une revue de littérature pertinente. Ainsi, la documentation étudiée permettra d'approfondir plusieurs sujets, tels que le contexte global des applications de géo-simulation, les automates cellulaires et le ou les rôles(s) joués par les grilles. Aussi, un survol des différentes grilles qui sont employées dans plusieurs domaines en général et leur usage dans les plateformes de simulation en particulier sera effectué tout en explorant leurs avantages et leurs inconvénients. Par ailleurs, divers travaux dans le domaine des SIG, de l'analyse de donnée spatiale et de la visualisation de l'information seront également étudiées. Ensuite, il

faudra penser à concevoir de manière incrémentale un modèle de classe selon un formalisme objet pour modéliser la structure de grille hiérarchique comme un premier noyau de base de notre système. Une fois que le modèle conceptuel sera réalisé, il est important de créer les principaux algorithmes employant la structure retenue selon une approche incrémentale et itérative. De plus, la construction et l'expérimentation d'un mécanisme d'indexation sera approfondi. Cette étape se fera selon une approche expérimentale par itération et adaptation (agile) avec la réalisation du prototype qui emploie ce genre de grilles. Finalement, d'autres itérations suivront pour aborder et solutionner à leurs tours les autres problématiques spécifiques que notre projet traite et par le fait même enrichir le premier prototype pour fournir à la fin, une application complète et fonctionnelle.

1.5 Présentation du mémoire

Dans le chapitre 2, nous présentons une revue de littérature générale des différents domaines liés à notre contexte de recherche tels que les automates cellulaires et les interfaces graphiques zoomables. Dans le chapitre 3, nous présentons une autre revue de littérature plus spécifique aux problématiques de notre projet. Ce chapitre représente notre état de l'art et il contient tous les fondements théoriques dont nous nous sommes inspirés pour réaliser notre projet. Le chapitre 4 représente notre solution pour créer des grilles hexagonales hiérarchiques. Quant au chapitre 5 il aborde tout l'aspect d'analyse et de préparation des donnés. Ce dernier présente également un phénomène urbain que nous avons déterminé et que nous avons utilisé à titre d'illustration de notre approche. Par ailleurs, le chapitre 6 aborde l'architecture de notre système, le lien entre les grilles et les automates cellulaires et les résultats obtenus. Finalement, le chapitre 7 présente une discussion des résultats obtenus, une conclusion générale pour l'ensemble du projet et les travaux futurs.

Chapitre 2 SIG, simulation et les données Géospatiales (État de l'art)

Ce chapitre présente des axes de recherches qui sont étroitement reliées aux objectifs de ce projet. Il représente une revue générale concernant les différents domaines que ce projet implique, tels que la géomatique, les automates cellulaires et le regroupement (clustering). En effet, par l'entremisse de ce chapitre, nous vison à nous familiariser avec ces domaines et ce, dans le but d'élaborer, par la suite, une revue de littérature moins générale et plus spécifique à nos objectifs de recherche.

2.1 Introduction

Ce chapitre représente les principaux champs de recherches liés à notre problématique. Les sections que nous abordons sont au carrefour des sciences géomatiques, de l'intelligence artificielle et de l'informatique. Nous allons présenter la géosimulation, les automates cellulaires et les interfaces graphiques pour des applications géographiques et finalement le Data mining spatial.

2.2 La géosimulation et automates cellulaires

2.2.1 La Géosimulation

2.2.1.1 Définition

C'est seulement depuis quelques années que la géosimulation (Benenson et al 2005) a pu devenir une discipline à part entière, à cause du progrès des technologies de l'information en général et de l'ingénierie du logiciel en particulier. En effet, le progrès dans les systèmes d'informations géographiques est le facteur crucial de son avancement. Ce genre de systèmes offre des outils et des méthodologies pour gérer, analyser et archiver l'information qu'elle produit. Aussi, d'autres facteurs issus d'autres sciences, telles que les sciences physiques et les mathématiques surtout, les automates introduits par Turing

(Wolfram S, 1984) ont influencé sa découverte. Par définition, la géosimulation (Benenson et al 2005) est un domaine, qui s'appuie sur la construction d'applications dans le but d'explorer des idées et des hypothèses des simulations géographiques. Ces applications sont basées sur des modèles complexes, car ils représentent des phénomènes dynamiques et réalistes qui se déroulent dans les milieux urbains. Ce sont ces propriétés qui la différencient des autres types de simulation.

La géosimulation se caractérise par :

- *La gestion des entités spatiales* : Les entités spatiales, comme les territoires, les maisons, les personnes et les zones géographiques, sont représentées sous la forme d'agrégats. Ces derniers peuvent être modifiés spatialement et partitionnés géographiquement.

- *La gestion des relations spatiales* : Les entités géographiques ont une ou plusieurs interactions les unes avec les autres. Cette dynamique est une représentation de haut niveau due aux comportements collaboratifs des objets géographiques à un plus bas niveau.
- *La gestion du temps* : La gestion du temps des objets est gérée soit de manière synchrone, c'est-à-dire ils évoluent tous en même temps, soit ils évoluent tous en des temps différents ou encore, de manière asynchrone.

2.2.1.2 Géosimulation et système multi-agents

La géosimulation ressemble à la modélisation orientée-agent (Benenson et al 2005). Par contre, elle en diffère en termes de traitement d'espace. Elle demande une représentation explicite de l'espace, en l'occurrence le comportement spatial des objets et leurs relations spatiales. Les systèmes multi-agents ou les « *MAS* » peuvent être vus comme des automates, car ils spécifient leurs états et spécialement les règles de transitions afin de donner une autonomie de comportement aux agents. Par exemple, un agent qui représente une personne peut avoir des états statiques (nombre d'enfants, salaire etc.) et des règles de transitions pour simuler le comportement humain. Il peut avoir également des états dynamiques (réaction aux congestions de trafic, voyages, faim etc.). Il peut avoir aussi d'autres catégories de règles pour gérer les relations spatiales telles que la navigation,

trouver les chemins, trouver les voisins, etc. Plus de détails au sujet des *« MAS »* et ce genre de règles seront abordés par la suite.

2.2.2 Les automates cellulaires

2.2.2.1 Historique des automates cellulaires

Les automates cellulaires ont été inventés par *Stanislaw Ulam* en 1984 (Torrens, 2003) et *John von Neumann* (1903-1957) (*Von Neumann, 1966*) à la fin des années 40 *au Los Alamos National Laboratory aux États-Unis*. En 1948, *Von Neumann* a proposé un article intitulé *«* Théorie générale et logique des automates *»*, lors d'une conférence tenue à Pasadena, en Californie. En 1949, il donna une série de cours sur le thème : Théorie et organisation des automates complexes. Une des questions centrales de ce cours était de savoir s'il était possible de concevoir une machine capable de s'auto-reproduire. Von Neumann émit l'idée qu'une machine est capable de manipuler des composantes d'autres machines élémentaires. Dans sa première conception, l'automate devait puiser dans un réservoir de composants de machine et construire un automate similaire à lui-même. Mais cela était inconcevable, étant donné l'état des technologies dans les années 50.

La solution à ce problème vint du collègue de *Von Neumann, Stanislaw Ulam* au laboratoire *de Los Alamos*. Ulam s'intéressait aux objets géométriques définis de façon récursive. Ces objets provenaient de jeux aux règles simples dans lesquels on pouvait voir des figures se développer, se reproduire, et combattre pour une portion de territoire et, finalement, mourir. Ces jeux se déroulaient dans un univers cellulaire, composé d'une matrice infinie où les cellules, régulièrement réparties, peuvent être dans un état passif ou bien un état actif. Les figures de cet univers étaient composées des cellules actives, et à tout moment, le devenir de chaque cellule était dicté par l'état des cellules voisines. *Ulam* s'aperçut à l'analyse de ces figures, qu'elles semblaient évoluer dans un monde qui leur était propre avec des lois bien spécifiques. Il suggéra alors à *Von Neumann* d'utiliser un tel monde abstrait pour pallier les difficultés technologiques qui se posaient pour la construction de l'automate auto-reproducteur. *Von Neumann* qui était habitué à voir les machines comme des circuits logiques, adopta l'univers d'*Ulam* pour commencer à réfléchir à son automate. Ce n'est qu'en 1966 que la publication du premier grand ouvrage consacré au problème de l'auto-reproduction voit le jour grâce à, *Arthur Burks*, qui

complète les travaux inachevés de *Von Neumann* et publie *Theory of self-reproducing automata*. En 1968, le second ouvrage consacré aux automates cellulaires est publié sous le titre *Cellular Automata*. Les années 60 voient aussi la résolution des premiers problèmes mathématiques liés aux AC. Au niveau pratique, les automates cellulaires s'appliquent dans des domaines aussi variés que la reconnaissance de formes, la cryptographie, l'étude du développement des systèmes urbains, etc.

2.2.2.2 Définition des automates cellulaires

Un automate est un ensemble d'éléments possédant des attributs qui changent au cours du temps (Benenson et al 2005). En adoptant ce regard, rien ne nous empêche de considérer que toute une ville pourrait être assimilée à un automate avec des états et des transitions. Un automate cellulaire est un système spatialement localisé, qui représente un arrangement d'automates individuels selon une forme précise comme par exemple une forme rectangulaire. Ils ont les mêmes spécifications que les automates traditionnels. Concrètement, chaque cellule représente un espace spatial discret et les limites d'un automate individuel. Chaque automate individuel a un voisin. Les voisins peuvent avoir plusieurs configurations, c'est-à-dire une agglomération de plusieurs cellules adjacentes définies par leurs distances à partir d'un automate individuel. Il y a deux types de configurations : *The nine-cell de Moor* et *The Five-cell de Von Newman*. Comme dans les automates traditionnels, des états sont affectés à chaque cellule. Dans les automates classiques, ces états évoluent au cours du temps et obéissent à des règles de transitions. Ce sont ces dernières qui déterminent la manière dont ces états vont s'adapter et évoluer au fil du temps. Ces règles peuvent être des instructions conditionnelles ou bien des opérateurs mathématiques.

2.2.3 Avantages et inconvénients des automates cellulaires

a) Les avantages

Malgré leur simplicité, les automates cellulaires peuvent représenter des systèmes complexes et offrent plusieurs avantages :

- *La décentralisation* : Les automates cellulaires et les « MAS » sont décentralisés. Il est indéniable que les automates sont individuels et indépendants. Ainsi, leurs interactions peuvent aboutir à une ou plusieurs actions collaboratives. Ces actions surgissent sans le contrôle d'une entité centrale. Le comportement de chaque automate individuel est régi par son état et par les informations qui existent dans son voisinage. Ces automates agissent indépendamment les uns des autres et chacun peut être actif à n'importe quel moment au cours du temps. La décentralisation de ces automates reflète la manière dont plusieurs systèmes complexes sont organisés comme les cités. Pour conclure, les automates cellulaires sont décentralisés; c'est pour cette raison qu'ils constituent un excellent mécanisme pour modéliser les systèmes réalistes.
- *Spécification nécessaire des détails* : Les modèles basés sur les automates cellulaires permettent une représentation détaillée des phénomènes à plusieurs niveaux de granularité. Ceci peut varier d'un phénomène lié à un territoire, jusqu'à ceux liés à une personne. Ils permettent aussi une description de ces élément via leurs attributs d'où la richesse de ce mécanisme. De plus, l'utilisation des automates cellulaires avec le paradigme orienté-objet, favorise, de manière naturelle et intuitive, l'encapsulation. Ce procédé permet une meilleure abstraction des spécifications des phénomènes urbains simulés.
- *Simplicité et Intuition :* Les automates cellulaires peuvent être construits de manière simple et intuitive en se basant sur la compréhension des actions des éléments.

b) Les inconvénients

- *Régularité des automates cellulaires :* L'inconvénient majeur des automates cellulaires est la définition des interactions spatiales de manière régulière dans un espace également régulier. Cependant, ce n'est pas tous les espaces urbains qui sont réguliers. Malgré cette contrainte, des automates cellulaires peuvent être implémentés avec une partition irrégulière, en utilisant, par exemple, les cellules de Voronoï (Gold, 2006). Le concept d'espace régulier et irrégulier sera élaboré en détail dans le chapitre 3.

- *Des cellules figées et incapables de se mouvoir :* Une autre faiblesse des automates cellulaires est l'incapacité de leurs cellules à se déplacer en dehors de leur voisinage. En d'autres termes, chaque cellule est incapable d'informer les autres cellules qui ne se situent pas dans son voisinage immédiat de son état ou de ses règles. Ceci est paradoxal dans un contexte d'application (la géographie urbaine), car ce dernier se préoccupe, des comportements et de la distribution des objets dans l'espace. Ces derniers peuvent être, des maisons, des territoires, des voitures, des personnes. etc. Ils changent de propriétés spatiales au cours du temps. S'il n'existe pas de mécanisme qui puisse doter les automates de la faculté de refléter le mouvement de ces objets et des relations spatiales entre eux, le degré du réalisme de la simulation pourrait diminuer grandement. Pour résoudre ce problème, Benenson utilise les systèmes muli-agents pour propager l'information indépendamment d'un voisinage donné. Nous présentons ce point en détails par la suite.

2.2.4 Systèmes géographiques à base d'automates (GAS)

2.2.4.1 Définition des GAS

Pour enlever les limites de l'incapacité de se mouvoir présentée précédemment, les GAS naquirent (Benenson et al 2005). En fait, les automates cellulaires sont combinés aux systèmes multi-agents afin de fournir un système entier pour la représentation des objets géographiques dans le cadre d'une géosimulation. Les GAS différent des automates traditionnels par l'ajout des propriétés qui tiennent compte de l'espace et des comportements spatiaux. Ils introduisent également un ensemble de règles géo-référencées pour situer les automates dans l'espace.

Ils définissent aussi des règles de voisinage et un ensemble de règles de mouvement qui est inspiré du concept de la mobilité issue des systèmes agents. Ces règles de mouvement permettent qu'un automate individuel ne soit plus encastré et figé dans son voisinage. Il peut ainsi mouvoir. Notons que par se mouvoir on sous-entend la capacité d'une cellule de véhiculer de l'information à une autre cellule ne se trouvant pas dans son voisinage immédiat. Grâce à ces règles, tout l'environnement géométrique est considéré comme un voisinage d'une cellule donnée.

Formellement, ceci est défini comme suit : ***G (K, (S, Ts) ,(L, ML) ,(N, Rn))*** où :

- *K* : l'ensemble des automates.
- *(S, Ts)* : l'ensemble des états *S*, associés à cet automate et pour chaque état, il existe un ensemble de règle de transition *Ts*.
- *(L, ML)* : L représente la localisation et elle est géo-référencée dans le système. *ML* dicte les règles de mouvement de cet automate. Ces derniers dépendent de l'input des données I des cellules voisines.
- *(N, Rn)* : Les règles Rn qui s'applique au voisinage *N*.
-

2.2.4.2 Types d'automates géographiques

Il existe deux types de GAS. On peut distinguer des automates géographiques fixes et non fixes. Les premiers représentent des objets qui ne changent pas de localisation durant le temps. Ceci est proche par analogie, aux cellules des automates cellulaires. Par exemple, plusieurs objets urbains peuvent être spécifiés avec cet automate : les liens entre les rues, les immeubles, les parcs, etc. Toutes les règles de transition peuvent leur être appliquées, à l'exception des règles de mouvements. Quant au deuxième type de GAS, ils sont à l'opposé des premiers. Ils représentent donc les entités qui changent de localisation durant le temps. Généralement, un système urbain fait cohabiter les automates fixes et les automates non fixes. Par exemple, un modèle de GAS, qui simule la dynamique des maisons peut être représenté par un GAS fixe avec des états qui décrivent des propriétés résidentielles (nombre de chambres, nombre d'étage, style d'architectures, etc.). Un GAS non fixe représentera alors des variables comme les revenus économiques de la famille, l'âge des parents et le nombre des enfants. Remarquons que les deux types d'automates dépendent l'un de l'autre. En somme, un milieu urbain peut être perçu comme une superposition de plusieurs couches dont celles de plus bas niveau sont composées de GAS fixes et représentent l'infrastructure urbaine. Quant aux couches supérieures, elles représentent les GAS non fixes ou encore les objets mobiles comme le montre la figure suivante :

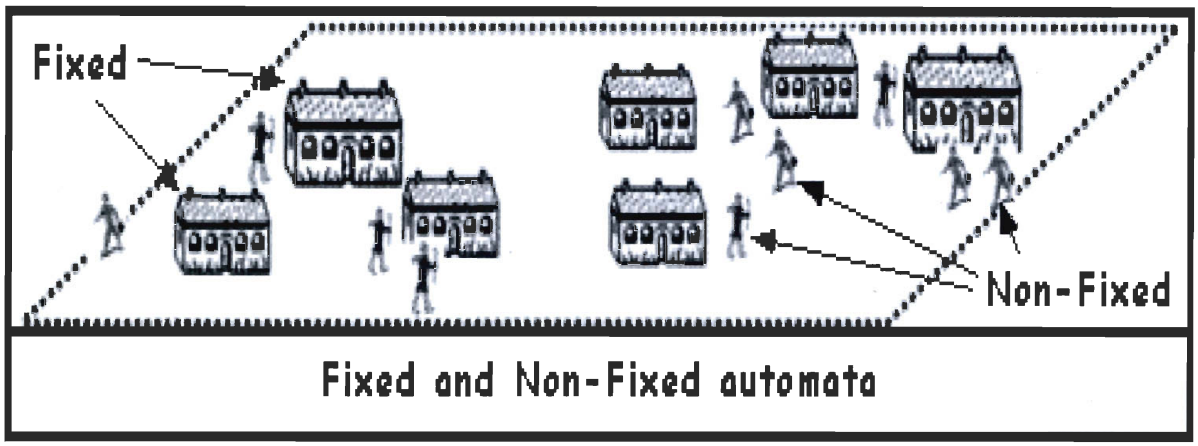


Figure 2.1 Automates fixes et mobiles (Benenson et al, 2005)

2.2.4.3 Les états des automates géographiques

Les variables d'états d'un automate fixe (définies formellement par le symbole S) décrivent des variables géographiques comme la hauteur, l'accessibilité, la visibilité etc.
À titre d'exemple, une maison est un automate fixe, car elle possède une accessibilité par le réseau de transport, une superficie géographique etc. Dans le cas des automates non fixes, elles décrivent des variables concernant le mouvement comme la destination, la vitesse, l'avancement vers la destination, etc. On peut citer comme exemple, une voiture qui se déplace avec une certaine vitesse vers une certaine destination.
Les transitions d'états (Ts) existent quel que soit le type d'automate. Il est important de dire que dans ce contexte, les automates cellulaires sont artificiellement proches des GAS, car les règles des transitions des états des cellules sont guidées seulement par ces cellules. Par contre, les états des objets d'infrastructures urbaines dépendent des objets de leurs voisinages. Ces derniers sont guidés par les agents des automates géographiques non fixes.

2.2.4.4 Référencement des automates géographiques

La convention de géo-référencement *L* montre comment les automates géographiques peuvent être enregistrés dans l'espace. Pour ceux qui sont fixes, le géo-référencement est statique, car les coordonnées de ces automates ne changent pas au cours du temps.
Généralement ce sont des références spatiales qui ont des valeurs constantes au cours du temps par exemple des cordonnées en *x* et *y,* d'où le nom géo-référencement statique.

En revanche, pour ceux qui ne sont pas fixes, le géo-référencement est dynamique. Par dynamique on sous-entend un géo-référencement qui n'est pas constant et qui change au cours du temps. En effet, du fait que ces automates peuvent se déplacer au cours du temps, leurs coordonnées changent fréquemment. Il est à remarquer que pour les automates fixes, il peut y avoir un géo-référencement dynamique, comme par exemple, le partitionnement du territoire lors de la simulation. On dit que les GAS peuvent être géoréférencés pour simuler l'espace directement et indirectement.

- ***Géoréférencement direct :***

  Les automates géographiques fixes sont toujours localisés en termes de géoréférencement direct. Ce dernier suit l'approche du modèle vecteur des SIG, car il utilise une liste de coordonnées indiquant les détails spatiaux nécessaires pour représenter les propriétés spatiales d'un automate comme par exemple ses frontières, ses nœuds, sa localisation etc.

- ***Géoréférencement indirect :***

C'est le fait qu'un automate pointe vers d'autres automates. Par exemple, on peut référencer les occupants d'une maison par une adresse. Le géoréférencement indirecte concerne les automates non fixes, mais il peut également s'appliquer aux automates fixes.

2.2.4.5 Règles de voisinage des GAS

L'ensemble des voisins *N* et leurs règles *Rn*, qui définissent les règles de changement des relations entre ces voisins, est une composante importante dans la définition formelle présentée précédemment. Pour les automates fixes, ces règles sont faciles à utiliser et à encoder, car les objets sont statiques dans l'espace (bien localisés et fixes). Ceci peut se faire via un réseau de nœuds ou via l'adjacence des unités dans les modèles réguliers et irréguliers. Concernant les automates non fixes, ils présentent plus de défi lors de la spécification des *Rn* parce que les objets et leurs relations avec le voisinage sont dynamiques dans l'espace et dans le temps. Ceci peut se faire en fonction de la distance ou des relations avec le voisin le plus proche. Cependant, d'un point de vue performance et vitesse d'exécution, ceci peut être lourd et coûteux surtout quand d'autres détails, tels que la visibilité ou l'accessibilité, sont impliqués. Dans ce cas, il est plus approprié de

modéliser les *Rn* en fonction d'une localisation indirecte en pointant directement vers leurs voisins.

2.2.4.6 GAS, utilisation et intérêts

Le système urbain est vu comme un système complexe. Les GAS dans ce contexte, sont construits comme une collection d'automates géographiques qui interagissent ensemble. Cette interaction peut être spécifiée de manière à faciliter l'émergence d'entités de haut niveau, de phénomènes et d'événements selon une démarche *bottom-up* (du plus bas niveau vers le plus haut niveau). De plus, les systèmes complexes montrent une organisation autonome d'émergence de phénomènes : d'où l'intérêt d'utiliser les GAS pour refléter cette dynamique urbaine. Par ailleurs, les systèmes spatiaux peuvent *« s'auto-organiser »*, car chaque type de GAS change ses propriétés pour favoriser l'émergence de nouveaux objets ou la dissolution d'autres, déjà existants. Cette émergence se fait de manière autonome.

2.2.4.7 La dynamique urbaine et les systèmes Multi-agents

Un *MAS* (Benenson et al 2005) est une collection d'agents ayant des relations entre eux et avec l'environnement dans lequel ils se situent. Également, les agents peuvent être interprétés en termes d'états et de règles de transition. En fait, la représentation d'une dynamique urbaine nécessite une variété d'agents ayant des états formulés de maintes manières. Parmi ces états, et pour respecter les facteurs spatio-temporels de la simulation, une unité temporelle est affectée aux agents dépendamment du contexte de simulation pour la rendre la plus réaliste possible. Aussi, afin d'animer ce genre de modèle, il faut spécifier les connaissances concernant les cités et la manière dont elles réagissent dans certaines situations. Des travaux dans le domaine des sciences sociales et psychologiques ont investigué le choix et le comportement des humains dans un contexte urbain.

Premièrement, la capacité de changer de localisation est une des plus importantes propriétés des activités des agents. Les agents urbains réagissent à des stimulations internes et à des stimulations d'environnement. Ces réactions sont « bi-dimentionnelles ». La 1^{ière} dimension se charge des changements des attributs non spatiaux des agents et, par analogie, de changer les états des cellules dans les automates cellulaires. La 2^{ième}

dimension représente les changements des attributs spatiaux, c'est-à-dire de localisation dans la cité.

Deuxièmement, la capacité de migration d'un automate agent est une des nouveautés introduites par les *MAS* géographiques permettant à la simulation de dépasser les limites des automates cellulaires à base de cellules fixes. La formalisation du comportement de migration est importante dans les *MAS* géographiques. Pour les agents, peu importe qu'ils soient fixes ou non fixes, proches ou loin, ils réagissent de la même manière avec les autres agents et les objets du modèle. Un outil de géosimulation à base d'agent nommé OBEUS (Benenson et al 2005), est développé comme une simple implémentation des GAS.

2.3 Les interfaces utilisateurs pour les applications Géographiques

2.3.1 Introduction

Le « zooming » (Bederson et al 1995) est un mécanisme efficace pour naviguer dans des données et des espaces énormes. Son implémentation et son cadre théorique sont adaptés selon le format des données. Par exemple, des données d'ordre documentaire, différentes de celles d'ordre 3D. À cet effet, et par rapport à notre problématique ou l'information est de type géographique, on s'est penché sur des techniques par lesquelles on peut représenter un environnement géographique « zoomable » et multi-résolution. Cette section est une percée qui vise à trouver la meilleure technique ou librairie utile à notre problématique.

2.3.2 PAD, PAD++ et Jazz

Bederson (Bederson et al 1995) présente les librairies PAD et PAD++. Elles permettent à l'utilisateur de naviguer à travers des données en utilisant le *« scrolling »* ou le *« zooming »*. Bederson a étudié également le *zoom sémantique*. En effectuant l'opération du *« zooming »,* certaines parties des données ont besoin d'être réévaluées. Lors de cette opération, des détails peuvent s'ajouter donnant naissance à une représentation tout à fait différente de l'objet initial, plutôt qu'il soit raffiné via un *« zoom in »*, ou bien abstrait par

le biais d'un *« zoom out »*. Dans ce cas, le zoom sémantique sert à cerner le contexte sémantique du *« zooming »* pour connaître ce que l'utilisateur cherche à représenter. Est-ce que ce sont les composantes hiérarchiques d'un objet qu'on veut afficher ou bien l'objet lui-même, mais à différentes tailles ? À titre d'exemple, l'auteur propose une horloge qui affiche initialement les heures et les minutes. À la suite d'un *« zoom in »*, on affiche les secondes et les dates et, inversement, lors du « zoom out ». PAD++ s'inspire des techniques de généralisation cartographique afin de créer des symboles alternatifs représentant l'information cachée aux niveaux d'échelles les plus bas. Ainsi, le contenu de la représentation graphique est différent d'un niveau d'échelle à un autre. Il remédie également aux limites d'efficacité cognitive du zoom et aux limites de performance de son prédécesseur PAD. Dans ce dernier, le zoom est désorienté, et par conséquent, il ne peut pas fournir une solution cognitive fiable pour la visualisation de l'information. PAD++ cède la place à Jazz (Bederson, et al. 2000), une interface zoomable avec Java. Plus récemment, une autre librairie en Java et C# a vu le jour sous le nom de *Piccolo.Net.*

2.3.3 « Continuous zoom »

Cette technique (Lyn Bartram et al 1995) gère un espace bidimensionnel rectangulaire en le partitionnant en des sous-rectangles également, créant ainsi une hiérarchie de rectangles composés. Les figures suivantes montrent l'organisation hiérarchique dans un tel système.

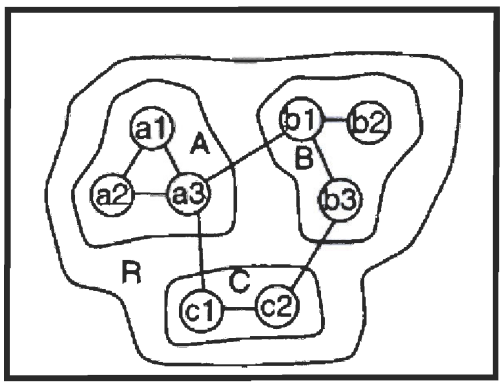


Figure 2.2 Système hiérarchique (Lyn Bartram et al 1995)

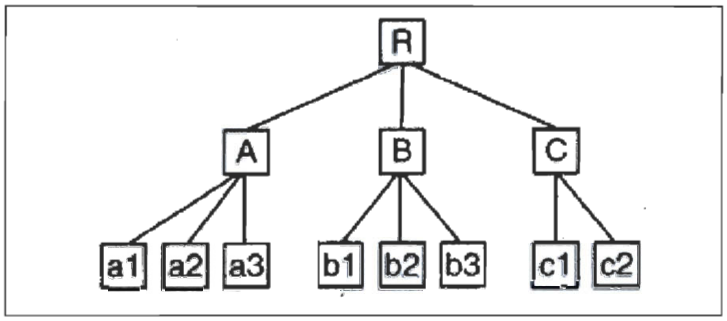


Figure 2.3 Arbre d'affichage de la hiérarchie (Lyn Bartram et al 1995)

Les nœuds à l'intérieur du système représentent les nœuds de l'arbre. Les nœuds internes qui ne sont autres, que des regroupements de nœuds sont des sous-arbres, aussi nommés des « clusters ». L'utilisateur contrôle le niveau de détail des zones d'affichage en ouvrant et fermant les clusters, permettant ainsi de voir la hiérarchie ou la profondeur des niveaux de détails dans l'arbre correspondant. À chaque opération, la taille des nœuds internes est ajustée. Comme la hiérarchie est visible tout le temps, les portions détaillées sont accessibles en tout moment.

2.3.4 « Orthozoom scroller »

Appert et Fekete (Appert et al 2006) présentent une composante graphique qui permet une navigation dans des données hiérarchiques à l'image d'une simple utilisation d'un « slider » ou d'un « scrollbar ». En effet, cette composante est une extension d'un « slider » classique, couplée à une technique de « zooming ». En somme, c'est en déplaçant la souris orthogonalement au « slider » que le zoom s'exécute.

2.3.5 « Flip zoom »

Le Flip zoom est un mécanisme de navigation qui pallie aux problèmes de pertes de contexte des interfaces zoomables. Il va sans dire que quand l'utilisateur quitte la vue globale (le contexte) et navigue dans un espace virtuel n'ayant aucun renseignement sur le contexte, il finit par oublier son parcours et se perd. Le flip zooming se base sur le principe des interfaces « focus et contexte». Ce principe est une solution aux limites de l'écran face à un grand volume des données, car il présente simultanément sur l'écran une information détaillée (le focus) et une autre plus générale toujours disponible et visible : c'est le contexte. En bref, le focus est une vue locale détaillée, alors que le contexte est une vue

plus globale montrant l'ensemble des données. Comme on le voit sur la figure 2.4, tous les objets sont présents dans la vue initiale sous forme de pages (1 à 20). Lorsque l'utilisateur zoome sur la zone d'intérêt (ici n° 10), l'espace se réorganise de manière à accorder au focus l'espace optimal tout en maintenant le reste de l'information disponible (Holmquist, 1997). Cette technique est mieux adaptée aux données de types documentaires.

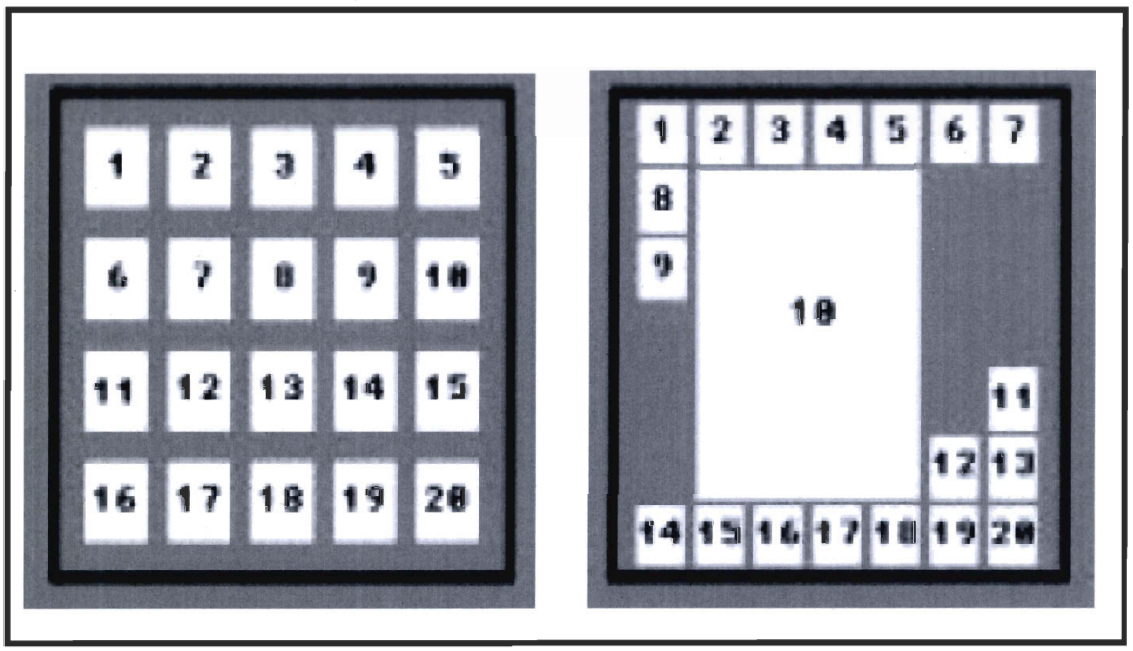


Figure 2.4 Flip Zooming (Holmquist, 1997)

2.3.6 Lentilles magiques

Les lentilles magiques appartiennent à la famille des interfaces de types focus et contexte, excepté le fait que l'information soit filtrée au lieu qu'elle soit déformée. En fait, la déformation est la modification de la taille des objets pour créer un point d'intérêt pour l'utilisateur (focus). Furnas (Furnas, 1986) permet de donner, à partir d'un élément, une valeur qui indique le niveau d'intérêt de cet élément. En général, à part les interfaces zoomables, les autres techniques existantes sont des techniques de distorsion. Cependant, les lentilles magiques modifient la représentation de l'objet, mais avec un filtre et non une distorsion. Ainsi, des filtres sont appliqués pour mettre l'accent sur des données spécifiques et ainsi omettre les informations qui ne sont pas pertinentes. (Bier et al 1993). Son principe est le suivant : la fenêtre représente une lentille sur la vue initiale. Les objets

sous cette fenêtre sont filtrés, soit en les présentant d'une manière différente sans modifier leurs rapports spatiaux, soit en dévoilant leurs caractéristiques les plus fines. L'avantage majeur des lentilles magiques est l'utilisation du zoom tout en gardant le contexte toujours présent. Un autre avantage est la possibilité de combiner plusieurs lentilles sur une même zone. Ainsi, l'objet perçu est une combinaison de plusieurs filtres différents.

2.3.7 Conclusion

Nous avons présenté un survol des principales techniques de zooming. Les interfaces zoomables présentent les objets à différents niveaux sans déformer leurs tailles. Au départ nous avons pensé que le « continuous zoom » est une excellente piste à approfondir et à utiliser dans notre prototype. Cependant, après quelques tests et essais, nous avons opté pour la librairie Piccolo.NE, qui est open source et qui est crée dans la tradition des outils PAD et PAD++ présentés plus haut.

2.4 Le Data Mining spatial

Parmi nos objectifs de recherche, nous visons à appliquer des techniques de regroupement (clustering) aux cellules des grilles que nous visons à développer. Pour ce faire, nous avons introduit cette section afin de comprendre les enjeux du clustering et de ses algorithmes classiques dans un contexte de data mining. Pour conclure, cette section vise à fournir un bref résumé sur ce domaine et tenter de trouver un algorithme qui nous inspirera pour développer un mécanisme de regroupement de cellules.

2.4.1 Introduction

Le Data Mining spatial constitue un domaine de recherche à part entière. En effet, sa spécificité par rapport au Data Mining traditionnel est qu'il prend en compte les relations spatiales entre les objets (Ester et al 1998). Il permet de :

- Trouver des règles de classification ;
- Former des clusters d'objets similaires ;
- Trouver des associations et des dépendances pour caractériser les données ;
- Extraire des tendances et de détecter les déviations.

Les méthodes permettant de réaliser ces tâches sont issues du domaine des statistiques et de celui des bases de données (Ester et al 1998). Le point commun de ces méthodes est l'exploitation des relations spatiales de voisinage. Ceci ajoute un niveau de complexité dans leur implémentation (Ester et al 1998).

Le Data Mining spatial est défini comme l'extraction de connaissances implicites, de relations spatiales ou d'autres propriétés non explicitement stockées dans la base de données spatiales (Han, et al 1997). Un de ses avantages est son aspect exploratoire pour extraire des connaissances à partir des données. De plus, les informations spatiales telles que la localisation et les relations de voisinage sont complètement intégrées. Ces dernières jouent un rôle primordial dans l'analyse spatiale et la découverte de connaissances (Han, et al 1997).

Le Data Mining spatial permet soit de découvrir des écarts mettant en évidences des spécificités locales, soit de chercher des regroupements de données (clusters). Cette phase permet de guider la phase décisionnelle où l'on procède à une analyse plus fine afin d'expliquer les écarts ou de caractériser les groupes à l'aide des méthodes de caractérisation, de règles de classement ou d'associations (Han, et al 1997). On s'intéresse tout particulièrement à l'aspect « clustering » de la première phase. Dans ce qui suit, nous présentons quelques algorithmes de regroupement.

2.4.2 Les Algorithmes de Clustering

Le « clustering » est une méthode de classification automatique non supervisée qui regroupe des objets dans des classes. Son but est de maximiser la similarité intra-classe et de minimiser la similarité inter-classe. (Sander, et al 1998). Ainsi, les objets appartenant à un même cluster sont plus similaires entre eux et dissimilaires aux autres objets contenus dans d'autres clusters. Cette technique est couramment utilisée en Data Mining et elle est bien connue dans le domaine des statistiques.

- Il existe un certain nombre de problèmes avec le regroupement, comme par exemple traiter un grand nombre de données n'est pas nécessairement rapide.
- L'efficacité de la méthode dépend de la définition de « distance », qui n'est pas tout le temps évidente à définir, surtout dans un espace multidimensionnel.

- Le résultat de l'algorithme de « clustering » peut être interprété de différentes façons.

À cause de ces problèmes, la plupart des recherches actuelles tendent vers l'optimisation des algorithmes de « clustering » afin de les rendre plus rapides dans un contexte à dominante spatiale. Dans ce qui suit, nous allons présenter les techniques classiques les plus répandues.

2.4.2.1 Algorithme K-means

Le K-means (MacQueen, 1967) est un des algorithmes d'apprentissage les plus simples sans supervision. Il résout le problème de regroupement. Il vise à classer un ensemble de données à travers un certain nombre de k clusters. Le nombre k est fixé a priori. L'idée principale est de définir pour chaque cluster un centroïde. La prochaine étape consiste à prendre chaque point appartenant à un ensemble de données et de l'associer au centroïde le plus proche. Ensuite, il faut calculer de nouveau chaque centroïde de chacun des groupes résultant de l'étape précédente. Finalement, les deux étapes précédentes sont répétées jusqu'à ce que les centroïdes ne changent plus. La mesure de similarité dans cet algorithme est donnée par la formule suivante :

$$\left\| x_i^{(j)} - c_j \right\|^2$$

Avec xi un point, et cj un cluster

Les étapes de cet algorithme sont :

1. Placer K points dans l'espace représenté par les objets qui sont à regroupés.
2. Assigner chaque objet au cluster qui a le plus proche centroïde de cet objet.
3. Lorsque tous les objets ont été assignés, recalculer les positions des K centroïde.
4. Répéter les étapes 2 et 3 jusqu'à ce que les centroïde ne changent pas.

2.4.2.2 Algorithme Hiérarchique

Étant donné un ensemble de n éléments à regroupés, et une matrice de taille n * n de similarité, le processus de regroupement hiérarchique (Johnson, 1967) est le suivant:

- Assigner à chaque cluster un point.

- Pour toutes les paires possibles de cluster, calculer la distance entre les 2 clusters de la paire, trouver la paire la plus petite distance et les fusionner en un seul groupe, de sorte que le nombre total des groupes n soit égal à n-1.
- Calculer les distances de similitude, entre le nouveau cluster et chacun des anciens clusters.
- Répétez les étapes 2 et 3 jusqu'à ce que n = 0.

2.4.3 Conclusion

Le Data Mining spatial présente une spécificité importante par la prise en compte des relations spatiales. Les travaux dans le domaine concernent plutôt l'implémentation d'algorithmes spécifiques comme le clustering. Toutefois, dans le cas du domaine spatial, les coûts algorithmiques sont prohibitifs en raison de la complexité des opérateurs géométriques et du volume des données.

2.5 Conclusion

Dans ce chapitre, nous nous sommes familiarisés avec le domaine et les différents champs de recherches qui sont étroitement liés à notre problématique. Nous avons exploré les systèmes d'information géographiques et la simulation géographique. Nous avons également étudié les interfaces usager utilisées pour ce genre d'applications. Finalement, nous avons examiné un dernier axe de recherche à part entière, qui est le Data Mining spatial. Dans le chapitre suivant, nous allons nous concentrer sur les grilles et la manière de subdiviser un espace puisque ce sujet est au cœur même de notre problématique.

Chapitre 3 Grilles, structure de données et indexage (État de l'art)

3.1 Introduction

Dans ce chapitre, nous présentons un état de l'art au sujet des grilles et des structures de données hiérarchiques. Nous allons aborder les différents types des grilles qui possèdent des formes géométriques différentes, et ce pour abstraire l'espace. Nous allons également montrer quelques techniques d'indexage de ces grilles en vue d'adapter l'une d'entre elles à notre problématique. Également, ce chapitre présente deux cas réels d'utilisation, qui sont pertinents à notre domaine de recherche.

3.2 Grilles et structures de données hiérarchiques

3.2.1 Grilles et tessellation

Bell (Bell, 1983) a défini le terme tessellation ou bien grille comme une division d'un espace en des sections ou en des composantes plus petites et plus atomiques. Notons qu'une composante atomique est la plus petite unité de base qui compose une grille. Elle peut avoir plusieurs formes géométriques : Aussi on parle généralement de grilles régulières et de grilles irrégulières. Une grille est régulière lorsque chaque composante est un polygone régulier (Bell,1983). Il existe 81 types, mais seulement 11 ont des structures adjacentes différentes. Les trois les plus connues parmi les grilles régulières sont : les grilles rectangulaires, les grilles triangulaires et les grilles hexagonales. Bell (Bell, 1983) a défini des critères qualitatifs afin de cerner les avantages et les inconvénients de chaque type régulier. Ces métriques sont les suivantes :

- *Adjacence* : Deux composantes d'une grille sont considérées comme voisines si elles sont adjacentes via un côté ou un vertex (un point). Une grille est uniformément *adjacente* si les distances entre le centre d'une composante atomique et le reste de son voisinage est la même.

- *Circularité :* C'est la différence des surfaces de plus petit cercle circonscrit et du plus grand cercle inscrit.
- *Orientation :* Une grille a une orientation uniforme, l'ensemble de ses composantes a la même orientation.
- *Limite :* Si une grille hiérarchique possède des composantes à un niveau k+1 et que ces dernières ne sont pas similaires à celles des niveaux inférieurs, alors la hiérarchie est dite limitée. En revanche, elle est dite illimitée si chaque composante peut se diviser en des composantes plus petites.
- *Régularité :* Une grille est régulière si la composante de base ou atomique est un polygone régulier.
- *Démocratie :* Si on ne peut pas différencier entre une grille d'un niveau k+1 et celle d'un niveau k, alors il existe une démocratie.

3.2.2 Grilles irrégulières et triangulation

La différence essentielle entre les structures régulières et les structures irrégulières autre que la forme géométrique, est le fait qu'une unité irrégulière est construite à partir des données. La taille, la forme et l'orientation des cellules reflètent les distributions réelles des données elles-mêmes. Cela est utile pour une interprétation visuelle des données. Le principal avantage de ces structures est aussi leur principal inconvénient. Du fait qu'elles dépendent des données et reflètent leurs organisations, n'importe quel changement de ces données peut changer toute la structure. Il existe deux formes connues : les cellules de Voronoï (Gold, 2006) et les *triangulated irregular networks (TIN's)(Paris, 2007)*, ou encore les triangulations de Delaunay.

Les cellules Voronoï (Gold, 2006) sont calculées à partir des surfaces d'adjacence, qui sont égales entre les points distribués dans l'espace. C'est un cercle formé à partir des points éparpillés à travers le plan. La croissance de ce cercle continue jusqu'à atteindre les frontières d'un cercle intersecté avec un autre cercle ou bien les frontières du plan. Ces cellules sont utilisées pour le calcul d'adjacence et l'analyse d'approximation.

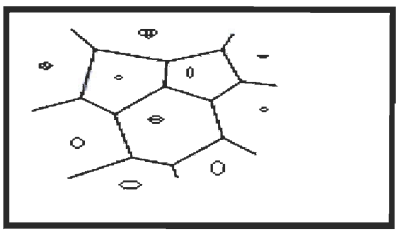


Figure 3.1 Cellule Voronoï

Les triangulations de Delaunay (Gold, 2006) sont le modèle standard de représentation des données d'un terrain pour des fins d'analyse et de visualisation. Ce modèle est construit à partir des points qui sont irrégulièrement distribués dans l'espace. Les facettes des triangles vont alors refléter l'aspect du terrain. Il représente le dual direct des cellules de *Voronoï*.

3.2.3 Grilles Régulières

3.2.3.1 Introduction

Dans cette section nous présentons une taxonomie des différentes grilles appartenant à la famille des grilles régulières. Nous présentons également leurs particularités et quelques contextes d'utilisation. Nous présentons essentiellement les grilles rectangulaires et hexagonales.

3.2.3.2 Grilles Rectangulaires et quadtree

Dans cette section, nous allons présenter à la fois les grilles et les structures de données hiérarchiques les plus utilisées avec ces gilles, car elles sont étroitement liées. En fait, le concept des structures de données hiérarchiques est utilisé dans les domaines de l'imagerie, les SIG et la robotique. Elles sont basées sur le principe d'une décomposition récursive, qui est similaire aux méthodes de diviser pour régner. Ce genre de structure est utile quand il s'agit de focaliser sur un sous-ensemble de données, en particulier parmi un ensemble entier. Plusieurs types de structures existent et offrent plus de performance en terme de temps d'exécution selon leurs contextes d'applications. Dans ce qui suit, nous allons présenter un survol de ces structures.

3.2.3.2.1 Quadtree par point

C'est une généralisation multi-dimensionnelle des arbres de recherches binaires (Finkel et Bentley 1974). Chaque point de donnée est un nœud dans un arbre ayant 4 sens, qui sont eux-mêmes des nœuds dans les sous-arbres correspondant aux quadrillage libellés *NE,NW,SE,SW*. Chaque point est supposé unique. Le principe d'insertion est analogue à celui des arbres binaires. En revanche, la suppression est plus complexe. Cette structure est intéressante quand il s'agit de faire de la recherche. Une requête typique est, par exemple, la détermination de tous les enregistrements dans une distance de 50 km d'un enregistrement donné en input. Par exemple, la recherche des villes situées à 50 km de *Washington D.C.* L'efficacité de cette structure réside dans le fait que plusieurs nœuds ne seront pas visités. Ceci est analogue aux arbres binaires. Bentley (Bentley et al, 1975) a analysé les opérations de recherche. L'exemple qu'il a traité est limité à deux dimensions seulement. En fait, le problème avec plusieurs dimensions est le facteur de branchement, c'est-à-dire le nombre de nœuds à la suite d'une division. Ce nombre devient de l'ordre de 2 à la puissance k ou k est le nombre de dimensions. Par exemple, si on considère trois dimensions, ce nombre va devenir de l'ordre de 2 à la puissance 3, donc égale à 8. Par conséquent, chaque nœud aura 8 enfants.

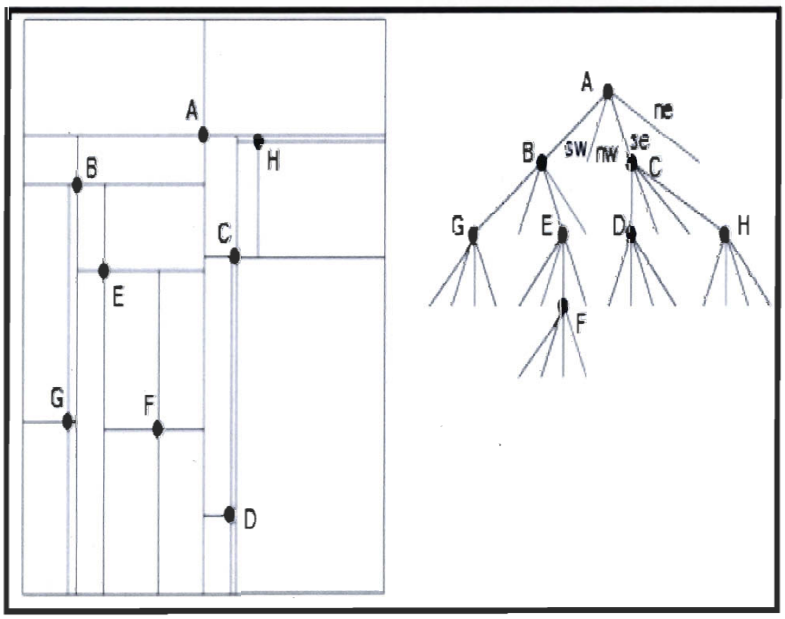


Figure 3.2 Point quadtree (Finkel et Bentley 1974)

Une variante du *point quadtree* est le *PR quadtree* (*point region quadtree*), qui à chaque fois, n'utilise pas un ensemble de points pour diviser l'espace, mais des rectangles en respectant le principe du *point quadtree*. Elle est utilisée en cartographie. En effet, une carte polygonale est une collection de polygones. Elle peut être représentée par le *PM quadtree*. Cette structure est efficace lors de plusieurs opérations comme : insertion de sommets des polygones, tester les points d'un polygone, superposer deux cartes, et finalement la recherche. Les efforts de la recherche au sujet de cette famille de structures, se sont orientés vers d'autres variantes et vers de nouveaux algorithmes afin de les manipuler de manière efficace. Par exemple, il y a le *CIF quadtree* qui est souhaitable pour les rectangles à l'image du *PM quadtree*.

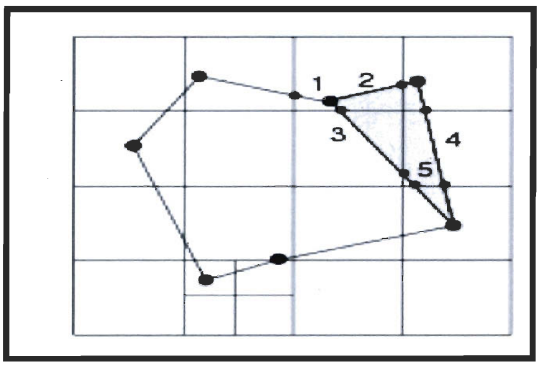


Figure 3.3 PM quadtree (Finkel et Bentley 1974)

3.2.3.2.2 Quadtree Linéaire

La décomposition de l'espace est importante lors de son partitionnement en des petites régions. Il y a plusieurs manières de le représenter. La plus répandue est le « quadtree » (Samet, 1984), qui est généralement implémenté sous forme d'arbre, dont chacun des nœuds pointe vers ses descendants (voir figure 3.4).

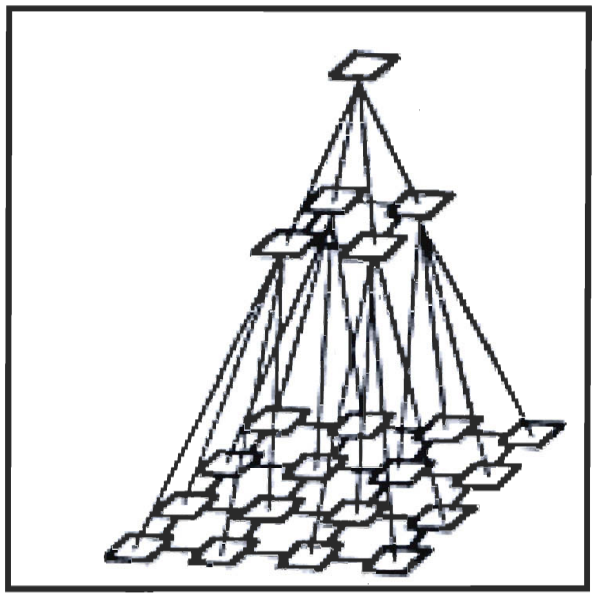


Figure 3.4 Quadtree Liniaire (Samet, 1984)

Une telle implémentation est un gaspillage de mémoire, d'où l'intérêt d'utiliser des variantes plus performantes, c'est-à-dire sans les pointeurs. La plus connue est le « *quadtree linéaire* » (Schrack, 1992). Elle est représentée par une collection de nombres correspondant aux nœuds. En fait, chaque nœud est représenté par une paire de nombres uniques. Le premier est l'emplacement du nœud dans l'arbre, le deuxième est le chemin parcouru. C'est le code de localisation (*location code*). Il est dit linéaire, car les bits du premier nombre sont concaténés et triés selon un ordre croissant qui reflète le chemin traversé dans l'arbre. Ce chemin est appelé « *path array*». Il utilise l'arithmétique binaire pour naviguer entre deux nœuds adjacents. Chaque nœud correspond à un rectangle régulier. Cette navigation est indépendante de l'emplacement des nœuds. En effet, Schrack (Schrack, 1992) y fait référence dans ses travaux sur la compression d'image avec des *« quadtrees linéaires »*. Rappelons que lors de la division d'un carré en des sous-carrés, chaque cadre est libellé de la manière suivante : *0: SW ; 1: SE ; 2: NW et 3 NE* (Samet, 1984). Ainsi, le code de localisation est formé en concaténant ces nombres à chaque feuille traversée de la racine jusqu'à une feuille particulière. Une fois ce nœud atteint, on affecte la valeur 0 aux bits les moins significatifs afin que leur nombre total soit égal à la résolution du domaine. Ce chiffre est associé au code de niveau, qui indique le niveau de la feuille à l'intérieur de l'arbre. Bref, une liste des codes de localisations, qui sont triés, est associée à

un code de niveau, pour former le *quadtree linéaire*. La séquence des codes de localisation est équivalente aux clés de Morton (Schrack, 1992), (Mark, et al 1990) quand ils sont sous la forme décimale. Ainsi, cette technique est plus efficace que le quadtree régulier en terme de temps d'exécution pour la recherche de voisinage, car le parcours de tout le chemin dans l'arbre est évité et remplacé par des opérations de manipulation de bits sur les codes de localisation. Ceci permet une performance accrue.

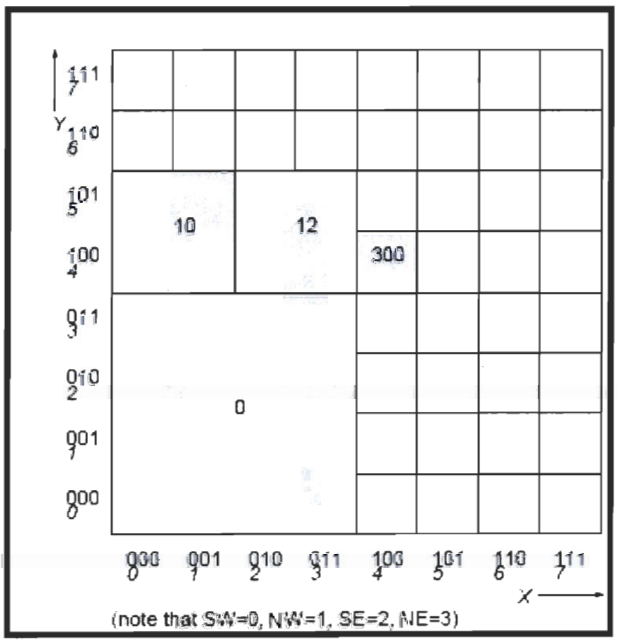


Figure 3.5 Quadtree Liniaire (Schrack, 1992)

3.2.3.2 Grilles Triangulaires

Samet (Samet et al, 1998) montre comment les travaux de Schrck (Schrack, 1992) sont adaptés à la décomposition à base de triangles équilatéraux au lieu de rectangles, pour que les opérations de navigation et de recherche soient effectuées de manière efficace indépendamment du chemin des nœuds. Bien que la solution proposée soit seulement appliquée à des triangles de même taille, elle montre aussi que la navigation entre des triangles de tailles supérieures ou égales s'effectue avec un temps d'exécution légèrement plus grand. Cette solution est à l'antipode d'une autre, proposée par Fekete, dont le temps d'exécution est toujours proportionnel au nombre maximal du niveau de décomposition (profondeur maximale du quadtree). C'est lors de la modélisation du globe terrestre avec un icosaèdre avec 20 faces triangulaires que Samet a utilisé cette technique. Néanmoins, cette

approche n'est pas limitée aux icosaèdres et elle est appliquée aux octaèdres et tétraèdres. Samet (Samet et al 1992) utilise une nouvelle méthode de numérotation tout en adaptant les techniques traditionnelles appliquées aux quadtrees. Par rapport aux études existantes, cette numérotation est similaire à celle proposée par Goodchild (Goodchild et al, 1992) pour l'octaèdre à l'exception de la méthode de recherche de voisinage qui est plus lente en termes de temps d'exécution. Également, la numérotation de Samet diffère de celle de Fekete (Fekete, 1990), Dutton (Dutton, 1996) et Otoo (Otoo et al 1993), au niveau des méthodes de recherche de voisinage. En effet, Dutton et Fekete divisent l'arête du triangle parent lors de la décomposition par bissection. Ceci engendre un schéma d'adressage dit flottant, opposé à celui utilisé par Samet et Goodchild qui est fixe.

Samet, Fekete, Dutton et Goodchild utilisent une division en 4 d'un triangle équilatéral. D'où le lien avec le quadtree traditionnel. Cependant, une autre approche serait de diviser un triangle rectangle en deux (bissection) (Floriani et al 1995, 2002). Ainsi, partant d'une grille rectangulaire, on divise récursivement un carré pour obtenir deux triangles rectangles. Ces deux derniers sont divisés à leurs tours en d'autres triangles rectangles. La récursivité est utilisée pour obtenir plusieurs enfants et par conséquent, plusieurs niveaux hiérarchiques. La structure qui supporte ceci est appelée « triangle bintree » dans laquelle chacun des deux nœuds fils sont les deux triangles rectangles résultant de la bissection du triangle parent.

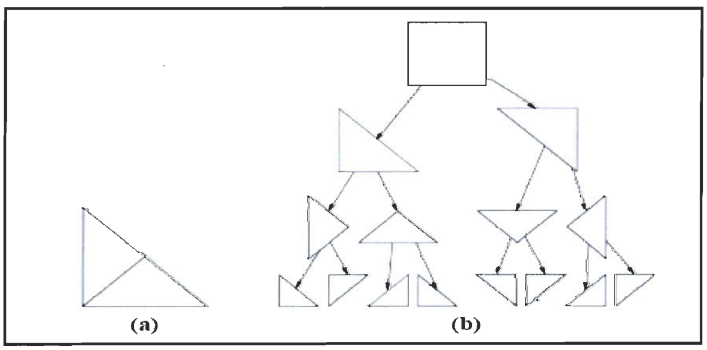


Figure 3.6 Bissection et triangular bintree (Floriani et al 1995, 2002)

3.2.3.3 Grilles Hexagonales

Cette grille a pour avantage majeur l'équidistance des points des centres de tous les voisins. En revanche, son inconvénient est le fait qu'un hexagone ne peut pas se diviser récursivement. Contrairement aux grilles rectangulaires, les cellules d'un hexagone ne sont pas faciles à adresser selon un système de coordonnées à base d'entiers, car les points ne sont pas alignés selon deux directions orthogonales. En raison de la propriété de symétrie que cette grille offre, une alternative consiste à utiliser, au lieu d'un repère orthogonal, un repère dont les axes est ceux de la symétrie de l'hexagone. Malheureusement, ceci n'est pas efficace, puisqu'il y a plusieurs axes à travers toute la grille. La manière la plus simple est donc d'utiliser deux axes asymétriques alignés selon les axes de symétrie de l'hexagone. Ainsi, on peut utiliser des coordonnées entières comme s'il s'agissait d'un plan (voir figure 3.7).

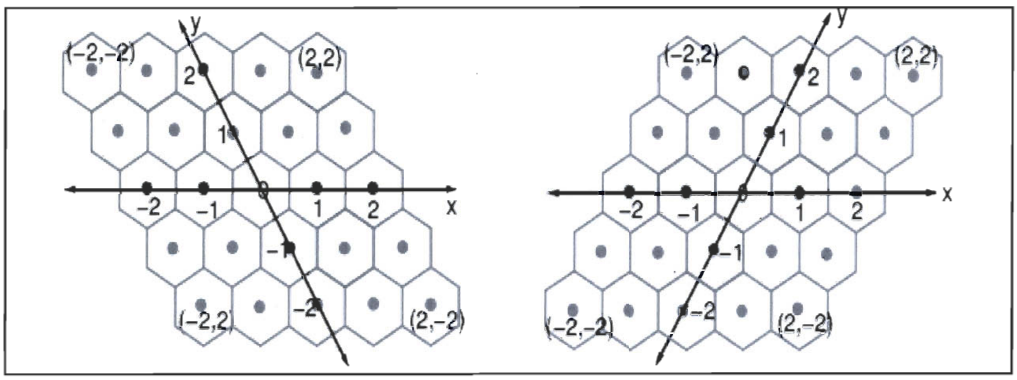


Figure 3.7 Axes asymétriques des grilles hexagonales (Burt, 1980)

Un autre procédé consiste à utiliser les trois axes de symétries d'un hexagone au lieu de deux axes. Le troisième axe est une combinaison linéaire des deux autres, d'où de la redondance. Cependant, ce procédé a l'avantage d'être efficace quand il s'agit d'opérations de rotation et de mesures de distance. Une autre méthode d'adressage est présentée dans la thèse de Burt (Burt, 1980). Il a examiné la possibilité de construction des pyramides hiérarchiques hexagonales et il a utilisé une structure de données appelée « sept-tree ».

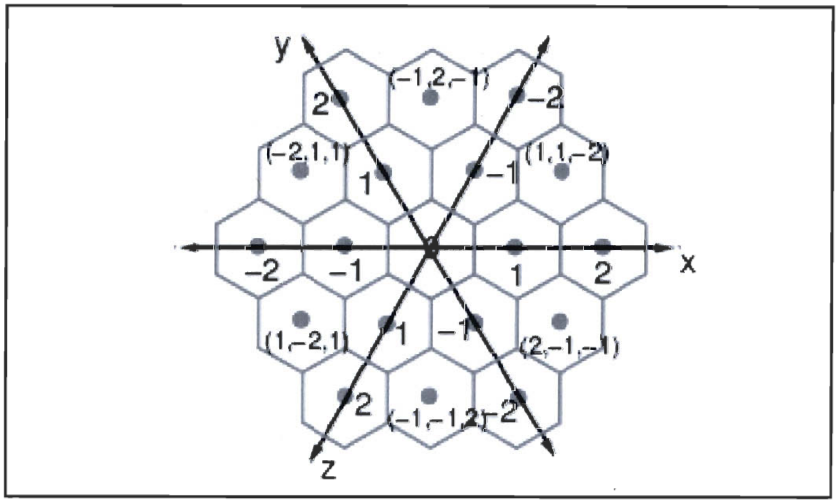


Figure 3.8 Adressage en utilisant trois axes asymétriques (Burt, 1980)

Également, Gibson (Gibson, 1982) a mis en place un mécanisme d'adressage qui tient compte des grilles hexagonales pyramidales ou hiérarchiques. Sa structure est extensible en plusieurs dimensions. En mode 2-D, elle ressemblait au « sept-tree » de Burt (Burt, 1980), mais l'adressage est différent. Sa structure utilisée est la GBT « *Generalised Balanced Ternary* » (Gibson, 1982). C'est une structure hiérarchique dont un niveau est formé par des agrégations des cellules des bas niveaux inférieurs qui utilisent des règles de regroupement. Dans le cas d'un mode bidimensionnel, ces règles sont identiques à celle de Burt (Burt,1980) où chaque niveau est une composition de type « cluster » de 7 cellules. Dans cette structure, le système d'adressage est le suivant : la cellule centrale a pour adresse 0. Les 6 cellules voisines sont numérotées dans le sens des aiguilles d'une montre. Chaque hexagone a une adresse unique représentée par une séquence de nombres. Chaque nombre de cette séquence correspond au niveau d'agrégation. Par exemple, la séquence 536 correspond à l'hexagone à la 6 ^{ième} position du premier niveau, qui se trouve à la 3 ^{ième} position du deuxième niveau, celui-ci étant à la $5^{ième}$ position du $3^{ième}$ niveau. Ce mécanisme d'adressage possède une arithmétique qui permet aux adresses d'être ajoutées, soustraites et multipliées. En fait, ces opérations correspondent aux opérations géométriques des vecteurs d'un plan. Indépendamment de cette structure, d'autres chercheurs (Hartman et Tanimoto, 1984) ont exploré la génération de pyramides hexagonales pour la modélisation cognitive de la navigation et de la localisation dans les cellules. La structure utilisée est basée sur une agrégation de triangles qui représentent des

pixels issus d'une image de forme hexagonale. Chaque pixel est libellé par un unique nombre de coordonnées formé par trois chiffres. Le premier indique le niveau dans la pyramide et les deux autres indiquent la position à l'intérieur de ce niveau. Bell (Bell et al 1983) évoque le principal inconvénient de la grille hexagonale qui est sa limitation lors de la décomposition. Il mentionne que cette limitation réside dans le fait que la forme hexagonale n'est pas illimitée. Un hexagone ne peut pas se décomposer en d'autres hexagones infiniment, contrairement à la forme rectangulaire. Pour lever cette limite, Bell (Bell et al 1989) a introduit la structure HoR *« Hexagonal or Rhombus »* (Bell et al 1989). Dans cette structure, il y a une fusion des centres des 4 hexagones adjacents pour former des rhomboïdes. Ainsi, cette structure formera des formes géométriques identiques et décomposables de manière illimitée. Par identique on sous-entend qu'il n'y a aucune différence de forme entre les enfants et les parents.

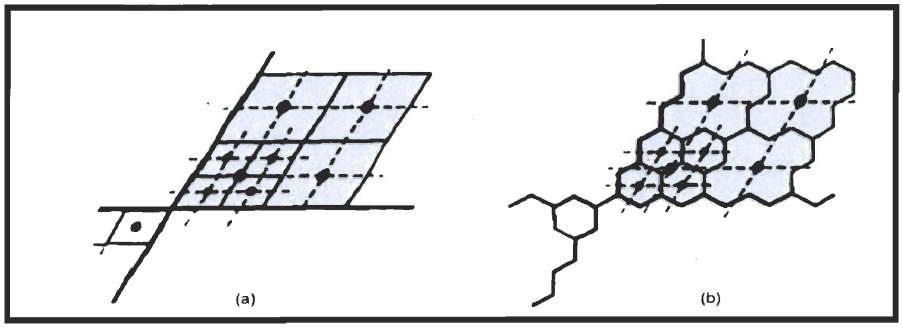


Figure 3.9 HoR et fusion des centres des hexagones (Bell et al 1989)

Cependant, elle présente un inconvénient majeur. C'est celui d'avoir l'illusion optique d'Orbison (Robinson 1972). En effet, lors de l'affichage, les cellules semblent tordues.

On peut également penser à diviser la cellule hexagonale à l'infini, puisque chaque cellule hexagonale se décompose en des triangles équilatéraux et eux, à leur tour, en d'autres triangles équilatéraux. Cette piste nous a semblé intéressante pour le regroupement. Ceci sera abordé par la suite. La figure qui illustre cette approche est la suivante :

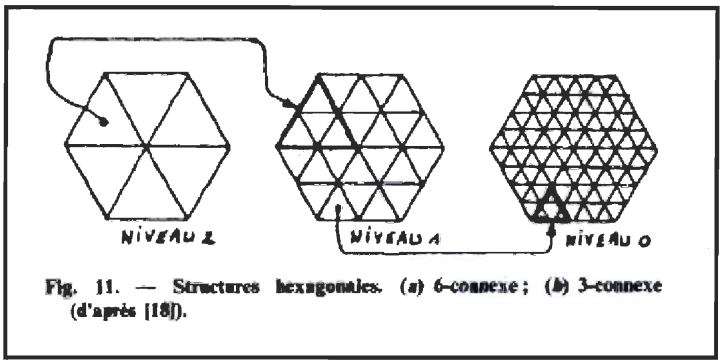


Fig. 11. — Structures hexagonales. (a) 6-connexe ; (b) 3-connexe (d'après [18]).

Figure 3.10 Décomposition d'un hexagone en des triangles (Rhind ,1981)

Cette figure nous montre comment un hexagone peut se décomposer en des triangles équilatéraux. En effet, à partir d'un grand triangle initial d'un $2^{ième}$ niveau, ce triangle se décompose en 4 sous-triangles de plus petite taille et qui se situent à un premier niveau. Pareillement, un triangle de ceux du niveau 1, peut se diviser à son tout en 4 sous-triangles qui se trouvent au niveau 0. Cette approche illustre ce que nous tenterons de faire dans ce projet.

3.2.4 Conclusion

Nous avons présenté les différentes familles des quadtrees et les différentes formes des grilles irrégulières et régulières. La figure suivante résume tout ce que nous avons abordé dans cette section.

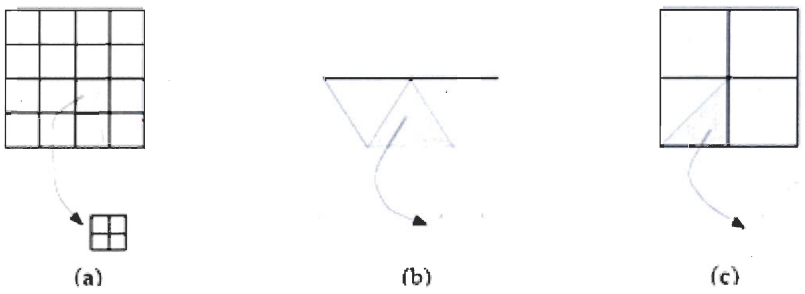


Figure 3.11 Récapitulative des différentes manières de diviser l'espace (Floriani et al 1995, 2002).

En effet, la composante a de cette figure nous montre les grilles carrés et les quadtree. La partie b nous montre comment un hexagone peut se décomposer en des triangle et cela constitue le cœur même de notre problématique. Finalement la partie c, montre une autre approche dans laquelle un carré peut être divisé en des triangles. Dans la section suivante, on s'intéresse aux grilles régulières et plus particulièrement, aux différentes manières d'indexer les cellules de ce genre de grille.

3.3 Indexage et recherche de voisinage

3.3.1 Introduction

Dans cette section, nous nous intéressons aux quadtrees et aux grilles triangulaires. Nous aborderons plus particulièrement les algorithmes qui traitent la construction des index et la recherche des voisins dans ces structures. D'un autre point de vue, cette section présente une synthèse des algorithmes à utiliser afin de tirer profit d'une synergie entre ces approches en les adaptant à notre problématique dans une solution hybride.

3.3.2 Courbes de remplissage de l'espace

Ce sont des courbes qui ordonnent des points dans un espace bidimensionnel. Ce sont ces courbes sur lesquelles se base l'indexage. Dans ce qui suit, nous présentons premièrement l'ordonnancement par ligne « *row ordering* » qui énumère les cellules ligne par ligne tout en énumérant, dans chaque cellule, les points de gauche à droite. Une variante est le « *row prime ordering* » dans laquelle chaque ligne est traversée dans le sens opposé. D'autres variantes considèrent les colonnes au lieu des lignes. Ce sont « *column ordering* » et « *column prime ordering* ». Ensuite nous présentons les clés de Morton (Morton keys) (Mark et Abel 1990) connues également sous le nom de « Peano-keys » ou « Z-Order » ou « N-Order » (Mark et Abel 1990), (Samet et al 1984), (Schrack, 1992). Leur fonctionnement se base sur la conversion de la valeur d'une ligne dans une base décimale en son équivalent dans une base binaire. Par exemple, la ligne numéro 2 aurait la valeur 10 en binaire et la ligne 3, la valeur 11 dans la base binaire. Par ailleurs, un autre mécanisme est le « *Gray code* ». Il a la propriété que des codes successifs sont différents seulement en une seule position de bit. Dans ce cas, quatre cellules voisines sont différentes seulement par un bit. Finalement, l'ordre de « *Hilbert* » est basé sur la courbe d'Hilbert-Peano. Il

existe un indexage qui suit une forme de spirale et un autre qui suit la courbe de « *Sierpinski* » basée sur une décomposition récursive des triangles. Cette dernière commence avec deux triangles qui sont divisés en deux, et ceci est répété jusqu'à ce que la résolution voulue soit obtenue. Toutefois, l'orientation et l'ordre des triangles sont importants. Une étude quantitative (Mark et Abel 1990) entre les séquences de Morton (Mark et Abel 1990),(Schrack, 1992) et la courbe d'Hilbert a montré qu'elles sont les plus efficaces, mais sans considérer les autres techniques présentées précédemment. Les différents types de courbes de remplissage de l'espace sont illustrés par la figure suivante :

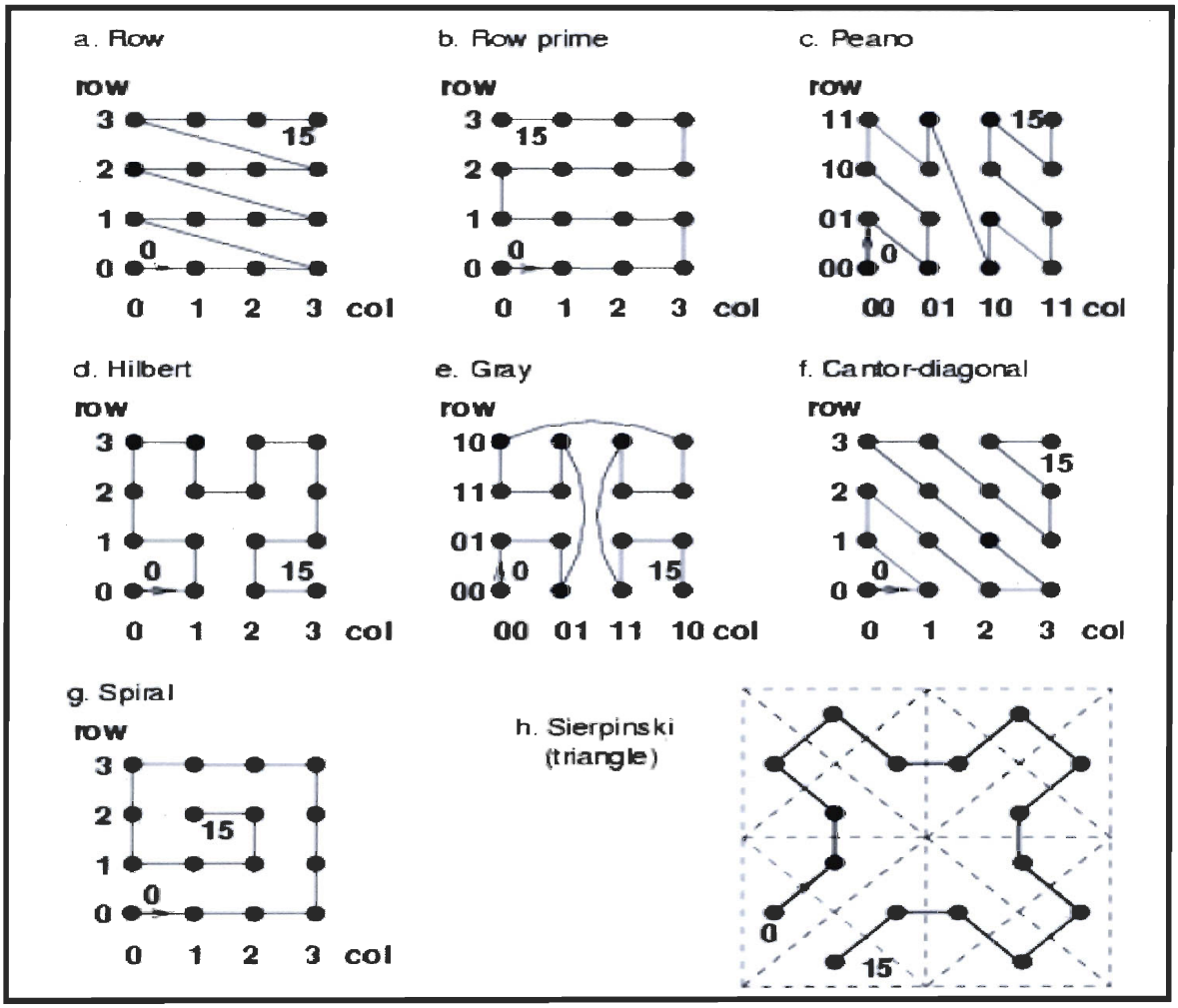


Figure 3.12 Différentes courbes de remplissage de l'espace (Rehind, 1981)

3.3.3 Indexage de grilles rectangulaire et triangulaires

Samet (Samet, 1999) a proposé un procédé pour indexer des grilles triangulaires, dans lesquelles chaque nœud est stocké dans un quadtree. Ainsi chaque cellule va représenter un triangle. Chaque triangle possède trois côtés et trois sommets (point ou vertex). Le sommet le plus haut de chaque triangle peut avoir deux sens d'orientation. Soit il est orienté vers le haut, soit il est orienté vers et bas. Chaque cellule possède une adresse ou un code. Ce dernier représente le chemin dans le quadtree « path array ». Ce code évolue à chaque décomposition d'une cellule parente en 4 sous-triangles enfants. En effet, chaque enfant rajoute 2 bits au « path array » en concaténant leurs adresses au code de localisation du parent. Ce n'est autre que le cumul des adresses à partir de la racine jusqu'à ce nœud sous forme d'une séquence de bits binaires. Cet indexage est appelé indexage flottant (Samet, 199).

Quant à la méthode de numérotation proposée par GoodChild (Goodchild et al, 1992), elle est opposée à celle de Samet, car son indexage est dit fixe. En fait, GoodChild attribue les chiffres décimaux 2, pour le triangle de gauche, 3 pour celui de droite, 0 pour celui du centre et 1 pour celui d'en haut ou d'en bas. Ceci permet une transition avec seulement des opérations mathématiques de soustractions et d'additions, et autorise un temps d'exécution constant (voir figures 3.13 et 3.14).

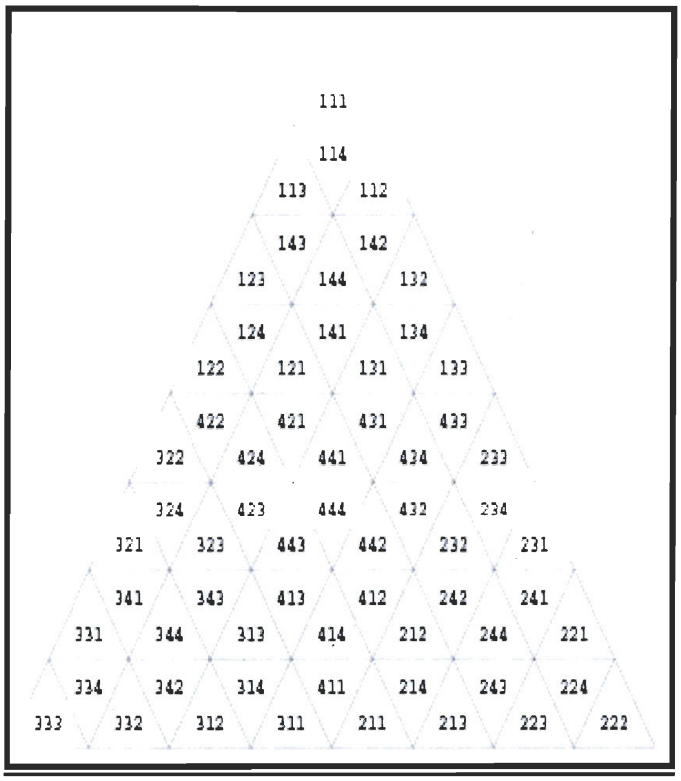


Figure 3.13 Indexage flottant de Samet (Samet, 1999

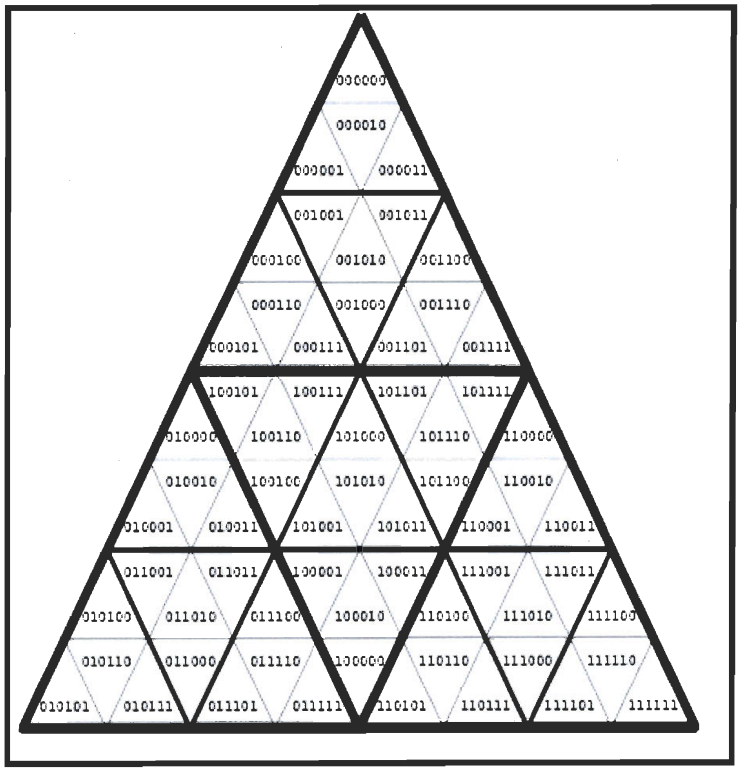


Figure 3.14 Indexage fixe de GoodChild Samet (Samet, 1999)

Nous avons également présenté l'approche de Floriani qui vise à décomposer par bissection une cellule carrée en des triangules rectangles (Floriani et al 1995, 2002). Il est indéniable que Floriani utilise un index fixe qui est similaire à celui de Samet. En effet, puisque le triangle rectangle est divisé en deux par bissection, un seul suffit pour coder l'adresse d'une cellule triangulaire dans l'arbre binaire triangulaire « triangular bintree », tandis que Samet utilise 2 bits par cellules puisqu'il y a une division en 4 enfants. Pour conclure, tout comme dans les arbres binaires, à chaque étape les bits 0 ou 1 sont ajoutés au code de localisation. Gestner (Gerstner, 2003) propose une implémentation de ces « bintrees » ou les adresses sont stockées dans un index externe sous forme d'un arbre B (B-tree).

3.3.4 Algorithmes de Recherche d'un voisinage

Samet (Samet, 1999) montre comment adapter des techniques classiques pour sa décomposition à base de triangles. Ensuite, elle présente son nouvel algorithme dont le temps d'exécution est constant. Dans ce qui suit, nous allons présenter les avantages et les inconvénients de chacun.

3.3.4.1 Algorithme de l'ancêtre commun le plus proche :

Il est basé sur une reconnaissance des patrons de bits pour une direction donnée, dans le code de localisation. Sa philosophie est la suivante : retrouver l'ancêtre commun le plus proche, c'est-à-dire l'ancêtre (cellule englobante) du niveau immédiatement supérieur au niveau d'intérêt. Ainsi, son adresse serait le code de localisation auquel on enlève les bits de la cellule de bas niveau. Par exemple, considérons le cas ou nous avons une adresse de 6 bits. Chaque couple de bits référence une cellule à cause de la division en quatre éléments. Notons que dans ce cas, notre résolution du domaine ou bien le nombre de niveaux est égal à 3. Si le code de cette dernière est 001100 on trouve que l'adresse de l'ancêtre commun est 0011.

Son fonctionnement est basé sur un *« scanning »* des codes de localisation jusqu'à trouver un patron donné, suivant qu'on se déplace à gauche, à droite ou à la verticale. Ceci se résume par une table d'états qui associe les patrons de bits, aux trois directions. Une fois cette adresse trouvée, la dernière phase vise à mettre à jour le code entier de la cellule

d'intérêt vers son voisin et ainsi le nouveau code à jour est l'adresse du voisin. Remarquons que le fait de remonter vers des cellules de plus haut niveau est l'intention de cet algorithme. Cette intention est analogue à l'approche dite de regroupement ou de « clustering ». Le seul inconvénient est sa complexité temporelle qui est de l'ordre du nombre de décompositions (profondeur de l'arbre). Par exemple si nous avons 3 niveaux, nous aurions une complexité de l'ordre de O (3) au pire cas (les bits se trouvent à la fin du code de localisation). Justement, c'est pour cette raison que l'algorithme de Samet a été proposé. Il permet un temps constant puisqu'il se réduit à des opérations arithmétiques entre les codes de localisation.

3.3.4.2 Algorithme à temps constant

Samet (Samet, 1999) se base sur la retenue (carry) de l'addition binaire. Le temps est constant si le code de localisation entre dans une instruction machine (32bit ou 64bit). Si plus d'un mot sont nécessaires, l'algorithme est d'une efficacité moindre, mais toujours à temps constant. La méthode est motivée par les travaux de Schrack (Schrack, 1992) sur les grilles rectangulaires. Pour les triangles, cela est différent, car il n'y a pas de corrélation entre leur décomposition et les coordonnées X et Y. En d'autres termes, étant donné que la grille de Schrak est rectangulaire, il est facile de l'associer et de l'indexer par rapport aux axes X et Y. Ceci ne s'applique pas aux grilles triangulaires, car il n'y pas de corrélation directe entre les coordonnées en X et en Y et l'index de la cellule, surtout au niveau des colonnes. Cependant, l'algorithme peut opérer de la même manière que celui de Schrak. Par exemple, pour transiter vers la droite, on se base sur la reconnaissance des patrons de bits. Ceci est facile quand les triangles sont frères (avec le même ancêtre). Cependant, quand il s'agit de deux triangles n'appartenant pas au même père, c'est-à-dire de 00 vers 01 ou 11 vers 00 on ajoute 1. Ceci ne s'applique pas au premier cas. Il faut remplacer toutes les séquences 00 par 11 afin que l'une des situations suivantes surgisse :

- Un carry est généré si nécessaire, alors le 00 est à la fin à droite du *« path array »*
- Un carry va être reçu proprement et les bits 00 sont les receveurs de ce *« carry »*.

Les deux cas sont gérés par l'addition. On remplace tous les 00 par des 11. En fait, on effectue l'addition binaire. Ainsi, tous les 00 affectés par l'addition deviennent 01, Quant aux autres bits 00, qui sont devenus 11 par l'addition, l'algorithme remet leur valeur à 00.

Pour les transitions à gauche, c'est la même chose qu'à droite, mais plutôt qu'avoir une addition, Samet (Same, 1999) utilise l'opération de soustraction binaire. Ainsi, au lieu de considérer la retenue de l'addition, elle considère le *« borrow »* de la soustraction. Finalement, pour les transitions verticales, Samet utilise le complément des deux derniers bits en se basant sur le principe de la symétrie par rapport à l'axe des X.

3.3.4.3 Évaluation des algorithmes

L'algorithme de l'ancêtre commun peut être raffiné facilement pour tenir compte du regroupement géométrique, c'est-à-dire fusionner les cellules de bas niveau en une de plus haut niveau et avoir ainsi une stratégie de regroupement fiable tant au niveau de la géométrie qu'au niveau de l'adressage, car l'adresse de la cellule agrégée est celle de l'ancêtre commun. Le seul inconvénient est sa complexité temporelle qui est de l'ordre du nombre de décompositions (profondeur de l'arbre). Par exemple, si nous avons trois niveaux, nous aurions une complexité de l'ordre de O(3) au pire cas (les bits se trouvent à la fin du code de localisation). Son avantage majeur est d'offrir un mécanisme naturel de regroupement facilitant les opérations de regroupement, de déplacement et de mises à jour des adresses. Si on implémente l'algorithme de Samet (Samet, 1999), il faut également un algorithme de regroupement, ou un effort plus considérable pour trouver l'adresse d'une cellule parente sans oublier l'opération d'unification géométrique, c'est-à-dire calculer les coordonnées du parent à partir des enfants. Cependant, pour l'algorithme du plus proche ancêtre, il permet avec peu d'effort de faire les deux en même temps en un seul algorithme. Rappelons que nous utiliserons un index externe pour stocker les adresses. Le seul mérite de cet algorithme est sa complexité temporelle puisque celle de l'ancêtre commun est relative à son nombre de niveaux. Or cette dernière est de l'ordre de quelques dizaines tout au plus, au pire cas. De plus, l'algorithme de Samet (Samet 1999) soufre du problème de lisibilité des adresses binaires. Nous avons établi une liste de critères qualitatifs pour classifier les avantages et les inconvénients de chaque algorithme.

	Algorithme Ancêtre commun	Algorithme Samet
Complexité temporelle	O (n), n nombre de décomposition	Constant
Complexité spatiale	Petite (3 listes + index externe)	Grande (plusieurs listes + plusieurs variables + index externe)
Effort pour Regroupement	Peu d'abstraction	Beaucoup plus d'abstraction
Implémentation	- 3 procédures courtes -Implémentation fonctionnelle.	- plusieurs procédures - Structure en couches -Couche pour fournir des services de bases (additions, soustractions, etc.) - couche de plus haut niveau pour utiliser ces services
Compréhension	Simple	complexe

Tableau 3-1 Comparaison entre les deux algorithmes

3.4 Cas d'utilisations

Quelles soient régulières ou irrégulières, les grilles ont été utilisées et appliquées à différentes fins et dans de nombreux domaines comme par exemple la modélisation de la surface de la terre, l'imagerie, la représentation 3D du relief d'un terrain, les bases de données spatiales, et la cartographie. Chacune de ces différentes approches utilise une structure de données interne pour sauvegarder les informations. Ces structures sont des adaptations des quadtrees ou bien des cellules de Voronoi dans le cas irrégulier. Aussi, un système d'indexage est-il utilisé pour référencer les éléments les uns par rapport aux autres. Nous pensons qu'étudier quelques approches pourrait nous inspirer lors de notre démarche. Nous avons étudié deux approches. L'une se base sur le quadtree et les grilles régulières (Paul Tsui et al, 2002), tandis que l'autre utilise des grilles irrégulières (Sébastien Paris, 2007).

3.4.1 Adaptive Recursive Tessellations (ART)

3.4.1.1 Introduction

Tsui (Tsui et al, 2002) a fourni le framework générique (ART) pour la formulation d'une série de structures hiérarchiques caractérisées par une multi-résolution à base de cellules rectangulaires régulières. La flexibilité de *ART* réside dans le fait de construire des

structures hiérarchiques de manière à ce que la taille et la forme des cellules soient variables à différents niveaux, et s'adaptent mieux aux besoins de l'application. Ce *framework* se base essentiellement sur le *quadtree* comme structure de données.

En fait, la plupart des études sur les *quadtrees* (Samet, 1984) mettent l'emphase sur sa capacité de compression des données; mais ceci n'est pas seulement la seule attraction que cette structure offre. En effet, elle a aussi l'avantage d'offrir une résolution variable pour sauvegarder des données géographiques (Samet, 1984) (Samet 1999) (Dutton, 1992). Cette résolution est gourmande en ressource et augmente le temps d'exécution, ce qui peut être critique dans des applications temps réels ou des applications interactives (Samet, 1992), (Schrack, 1992). Les *quadtree*s divisent la taille des cellules par 4. Cette division est traduite par la suite géométrique suivante:

$$s_i = a2^i \qquad i = 0, 1, 2, \ldots, n-1$$

- *si* : longueur d'un côté d'une cellule, au niveau i
- *a* : valeur par défaut de la longueur d'un côté
- *n* : nombre de niveau.

3.4.1.2 Définition de ART et de sa terminologie

Un modèle ART est composé premièrement d'un nombre de niveaux. Chaque niveau est une couche logique contenant une tessellation régulière selon une certaine taille de cellule ayant une certaine orientation. Plus le niveau est bas, plus la taille de la cellule est petite. Ceci reflète la résolution à échelle variable que ce modèle offre. La décomposition des tessellations de manière récursive se termine quand le niveau atomique ou encore le plus bas est atteint. Les niveaux sont notés de 0 jusqu'au niveau atomique. Théoriquement, le nombre de niveaux peut être infini, mais pratiquement il est fixé à cause des schémas d'adressage de la structure de données et des performances d'affichage. À chaque niveau, l'espace est régulièrement décomposé via une cellule unitaire appelée : unité de tessellation de base (BTU). À chaque niveau, la BTU a une forme rectangulaire et une longueur définie par les deux côtés adjacentes perpendiculaires. L'orientation est parallèle au repère initial. Finalement, les côtés des autres BTU, qui se situent à différents niveaux, doivent être alignées. Également, dans ART la géométrie de la tessellation est régulière, par contre, les paramètres de tessellation sont variables.

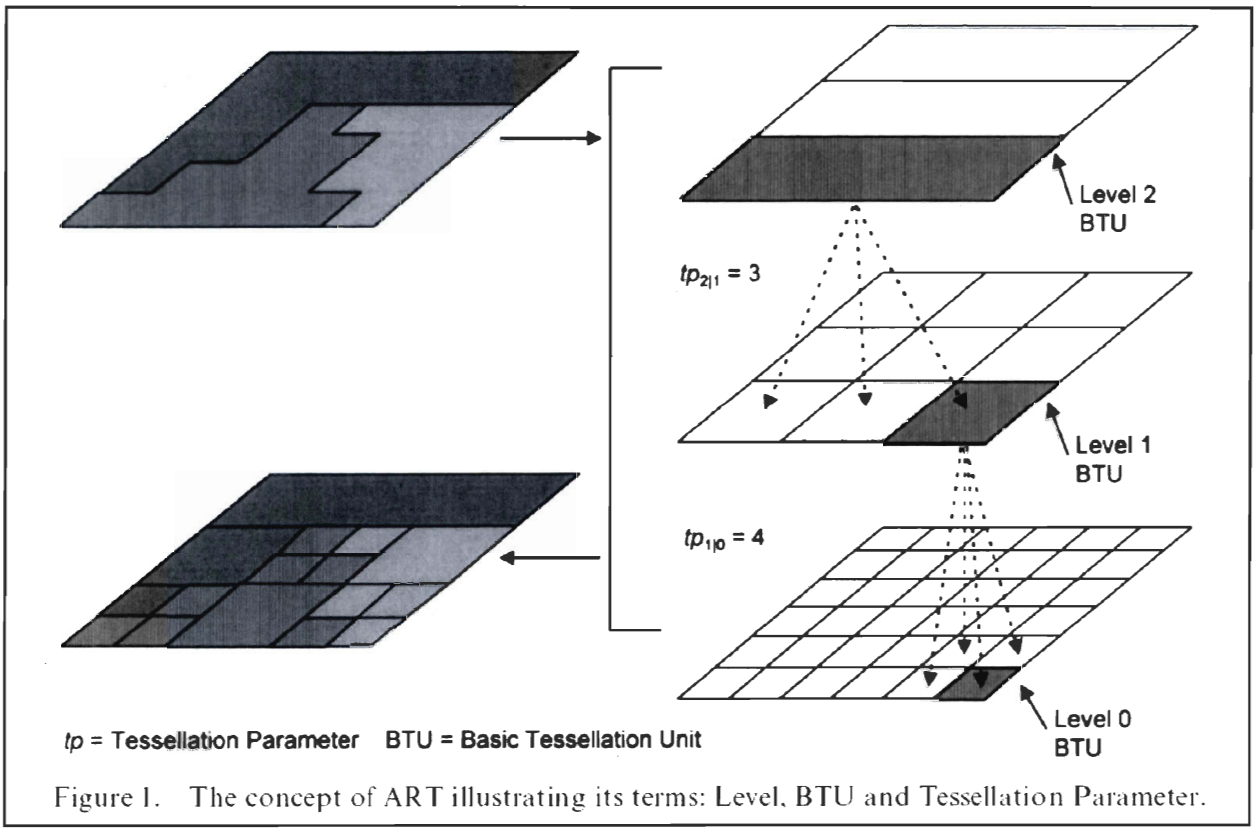


Figure 1. The concept of ART illustrating its terms: Level, BTU and Tessellation Parameter.

Figure 3.15 Illustration des niveaux hiérarchiques dans ART (Tsui et al 2002)

3.4.1.3 Conception du modèle de données dans ART

La philosophie de formulation du modèle de donnée ART est totalement différente des applications traditionnelles. En effet, pour ces dernières, la zone d'intérêt est identifiée en premier, ensuite les tailles des cellules sont déterminées. Finalement, l'utilisateur spécifie le niveau atomique pour déterminer la plus grande échelle de résolution qui peut être représentée. Contrairement à ceci, dans la construction d'un modèle ART, l'utilisateur détermine premièrement le nombre de niveaux et la taille des cellules basées sur les besoins de l'application.

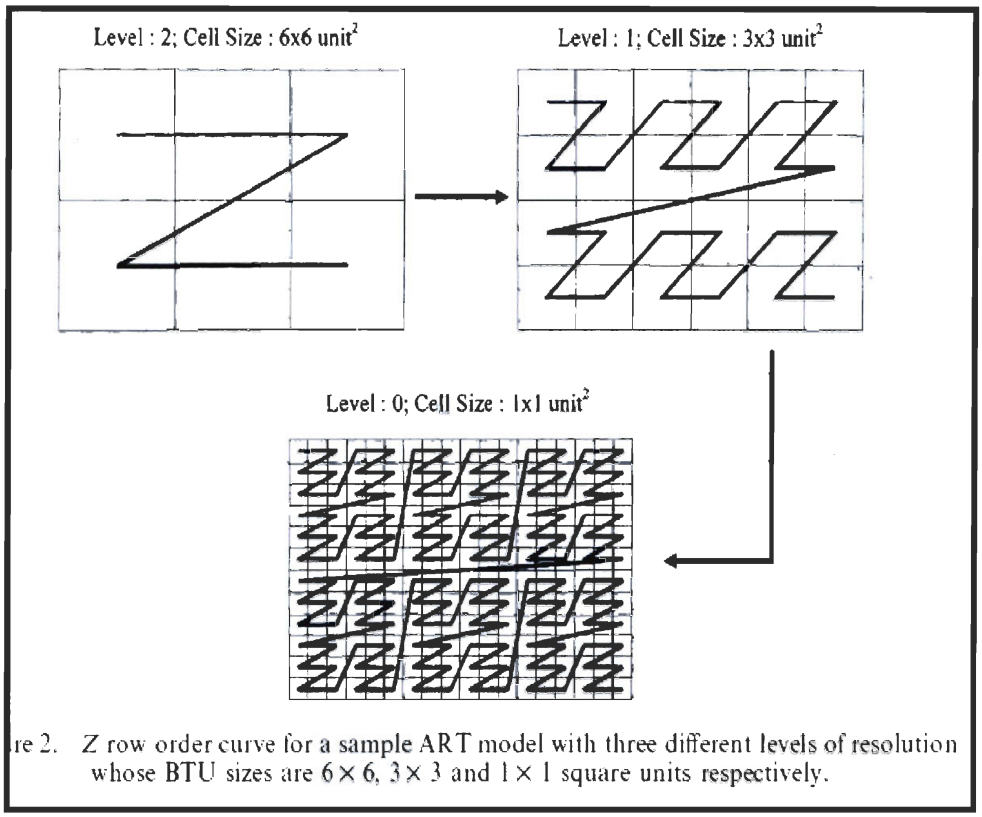
3.4.1.4 Structures de données dans ART

La structure de données de ART est dérivée de l'approche *two dimenstionnal run encoding (2DRE)* . C'est une technique de compression utilisée par les quadtrees (Lauzon

et al 1985). Cette dernière est formée par deux parties : La première consiste en un indexage selon les clés de Morton (Mark Abel et al 1990) et la deuxième est l'indexage et les structures des fichiers et des files d'attentes. Les séquences de Morton ont deux avantages dans le codage des *quadtrees* :

- La séquence de numérotation de Morton suit le pattern de décomposition des quadtrees. C'est-à-dire qu'elle offre un mécanisme naturel pour suivre l'ordre de décomposition des enfants.
- L'obtention d'un index particulier peut se faire par des opérations de bit des numéros des lignes et des colonnes d'une cellule. Ceci est identique aux travaux de Schrack (Schrack, 1992). Cependant, pour les quadtrees triangulaires, Samet (Samet, 1999) mentionne qu'il est difficile de trouver une corrélation entre les colonnes de la grille et les cellules.

La structure adaptative de ART est nommée *adaptative recursif run encoding ARRE*, qui a les deux mêmes propriétés citées plus haut, à l'exception du fait que le schéma d'indexage spatiale soit différent de celui de Morton.Dans la séquence de Morton (Mark Abel et al 1990) le codage de chaque cellule suit une courbe de remplissage de l'espace en Z. Cette courbe couvre les quatre côtés adjacents d'une cellule. ARRE applique le même chemin en Z à chaque niveau de haut en bas (top – bottum). Ce chemin numérote en premier la cellule la plus à gauche en haut et termine par la cellule la plus à droite en bas, comme le montre la figure suivante :



re 2. Z row order curve for a sample ART model with three different levels of resolution whose BTU sizes are 6 × 6, 3 × 3 and 1 × 1 square units respectively.

Figure 3.16 Indexation en Z dans la structure ART (Tsui et al 2002)

Cette structure affecte des décimaux consécutifs à chaque cellule atomique à partir de haut à gauche et en suivant un chemin en Z. L'intervalle de ces chiffres est 0, k-1 ou k représente le nombre total des cellules atomiques dans le modèle ART.

3.4.2 Une partition informée de l'espace pour la simulation de foule

3.4.2.1 Introduction

Paris (2007) découpe l'espace en des cellules de différentes formes pour reproduire un environnement identique à l'environnement réel qui n'est autre qu'une station de train, dans le but de créer une simulation de foule dans cette gare. Cependant, cet environnement est contraint, car il contient des obstacles: les murs. Cette contrainte a un impact sur la planification de la navigation des humains virtuels.

3.4.2.2 Environnement Virtuel et abstraction topologique

Le découpage de l'espace se fait grâce à la triangulation de Delaunay (La Marche, 2005) parce que chaque point de ce modèle est relié à son plus proche voisin via le côté du triangle auquel il appartient. Ainsi, lors d'une simulation, on peut facilement déterminer le graphe de voisinage des entités mobiles.

En effet, un triangle englobe la scène afin de représenter les frontières de l'environnement. Ensuite, la division est effectuée en tenant compte des obstacles à l'intérieur de la gare, produisant ainsi des cellules convexes qui ne sont autres que les lieux franchissables. Un graphe est extrait à partir de cette division dont les nœuds définissent les cellules et les arcs représentent les relations de voisinage. Ce processus produit des milliers de cellules. Pour des raisons de calcul, de performances et d'exploitation, ce graphe est inutilisable, d'où la nécessité de le conjuguer à une approche complémentaire pour le simplifier. C'est l'abstraction topologique. Elle permet de réduire le nombre d'éléments qui décrivent l'environnement et de proposer ainsi une définition plus intuitive des zones navigables. Ceci pour s'éloigner de la représentation géométrique et aller vers une représentation plus procédurale, tout en gardant les informations nécessaires. Cette abstraction est fondée sur des mécanismes de regroupement « clustering » : c'est donc une approche bottom-up. En effet, un nouveau graphe est construit à partir du graphe initial. Notons que le premier graphe initial, qui est issu des algorithmes de LaMarche (LaMarche, 2005) contient les informations géométriques. En fait, chaque cellule de haut niveau va contenir des cellules de plus bas niveau. D'où la notion de hiérarchie ou de graphe hiérarchique. Un parcours pour une traversée verticale est permis, et chaque cellule de chaque niveau doit connaître celle qui l'englobe ou qui la compose. Ainsi, plus le nombre de niveaux augmente, plus l'opération de descendre les hiérarchies pour aller chercher ces informations devient coûteuse. Aussi, l'expressivité de ce graphe va diminuer quand on augmente le nombre des niveaux.

Pour conclure, Paris introduit le concept d'abstraction topologique que nous avons défini plus haut afin de remédier au problème de l'augmentation du nombre de cellules et de l'exploitation du 1er graphe construit. En fait, à partir du graphe de subdivision initial, un autre graphe hiérarchique est construit. Ce dernier possède trois niveaux. D'où la notion d'abstraction. Cela permet donc, de réduire le nombre de cellules qui décrivent

l'environnement, et de produire une définition conceptuelle de l'environnement. À part le niveau initial (le graphe de décomposition), les deux autres sont les suivants :

- ***1^ier^ niveau d'abstraction :***

Du fait que la méthode TIN est sensible aux aléas géométriques (défauts d'alignements), un grand nombre de cellules est produit lors de la création du premier graphe, augmentant ainsi sa complexité. Le but est de grouper les cellules ayant la même sémantique. Ceci est basé sur deux heuristiques qui ont pour objectifs de maximiser les carrefours de l'environnement reproduit afin d'augmenter le nombre de nœuds de type carrefour. Cependant, ce genre de noeuds nécessite une prise de décision lors du parcours du graphe afin de permettre à l'agent de choisir vers quel voisin se diriger. Par conséquent, le temps alloué pour évaluer des opérations non décisives au niveau d'autres nœuds est réduit. Ces deux heuristiques sont appliquées l'une à la suite de l'autre.

- **2^ième^ niveau d'abstraction :**

Ce niveau cherche à construire des groupes de cellules pour caractériser des zones plus globales pour la navigation. Pour ce faire, on utilise les mêmes heuristiques que précédemment, sauf qu'elles agissent sur des groupes de cellules pour fabriquer des zones. Les heuristiques sont les mêmes, hormis celle de la convexité qui devient plus souple, car elle considère des groupes plutôt que des cellules. Ainsi, le même algorithme est utilisé avec une autre règle permettant le grossissement des carrefours pour devenir des zones.

3.4.2.3 Environnement Informé et nœuds conceptuels

Paris (Paris, 2007) a introduit dans son environnement géométrique des informations sémantiques qui sont utilisées pour la prise de décision des agents. Paris a utilisé les nœuds conceptuels qui permettent d'orienter le graphe et d'améliorer sa sémantique. Ce ne sont pas des nœuds physiques, mais des nœuds logiques. Ainsi, ils n'ont aucune surface et aucun impact sur la topologie. Comme ils peuvent être regroupés, ils vont servir à :

- Représenter plus précisément les entrées et les sorties de l'environnement en leur associant un sens de parcours.
- Connecter plusieurs graphes hiérarchiques.

Il existe trois types de nœuds conceptuels :

- *Bouchon :* Permet une modification dynamique des graphes ;
- *Nœud orienté :* Assigne un sens de parcours au graphe ;
- *Nœud d'interconnexion :* Connecte plusieurs graphes entre eux pour fabriquer un graphe de plus haut niveau.

Paris a également eu recours aux abstractions topologiques qui servent à réduire le nombre de cellules du graphe issu de la décomposition de LaMarche, vont aussi servir à informer l'environnement. Certes, elles vont contenir des informations relatives à la circulation des personnes. Ces connaissances ne sont autres que les facteurs influençant le mouvement des humains. Elles peuvent être de deux sortes, soit une information statique, qui n'est autre que la géométrie du groupe utilisée par les mécanismes de la simulation, soit les propriétés utiles à la navigation comme la distance et le vecteur de direction.

3.4 Conclusion

Dans ce chapitre nous avons présenté les principales approches dont nous nous sommes inspirées pour développer notre système, en particulier de Samet (Smaet, 1999) et de Schrak (Schrak, 1992) pour l'indexage et la recherche de voisinage. Nous allons également utiliser l'approche de Tsui (Tsui et al, 2002) ou celle de Paris (Paris, 2007) pour la génération de notre environnement.

Chapitre 4 : Grilles hiérarchiques hexagonales

4.1 Introduction

Rappelons que les cellules des grilles hexagonales ne peuvent pas se décomposer en d'autres cellules hexagonales de tailles inférieures (Bell, 1983). C'est dans cette optique que s'est inscrit l'objectif de notre recherche. En effet, nous tentons de palier à cette limite en travaillant sur des grilles dont l'unité géométrique de base est un triangle équilatéral. Ainsi, on pourrait construire plusieurs cellules hexagonales de différentes tailles à partir de 6 triangles équilatéraux et ce, sur plusieurs niveaux de résolution. Certes, en assurant une relation hiérarchique entre les triangles de petites tailles avec ceux de plus grandes tailles, on pourrait construire, à chaque niveau une cellule hexagonale à partir des triangles. Dès lors, il ne restera qu'à configurer les cellules hexagonales pour qu'elles ne se chevauchent pas et qu'elles aient la forme d'une grille hexagonale normale. Rappelons également que nous nous proposons d'avoir un mécanisme générique, c'est-à-dire de pouvoir paramétrer la construction de la grille selon l'usage et le contexte de l'application. Un exemple de paramètre est la taille d'une cellule ou encore, le nombre des niveaux hiérarchiques. Nous présentons, dans ce qui suit, notre démarche qui permettra d'atteindre ces objectifs. Le premier essai n'a pas été concluant, mais il a servi comme prototype qui aide à mieux comprendre nos besoins et les contraintes relatives à notre problématique. Ensuite, nous avons tenté un autre essai qui est réussi.

4.2 Essai non concluant

Cet essai est notre première tentative de résoudre notre problématique. Nous avons essayé de créer des fractales triangulaires afin de couvrir l'espace et ce, en modifiant l'algorithme de Sierpinski. Ensuite, nous avons établi des règles géométriques et d'indexage afin de cerner l'adjacence entre les différentes cellules de la grille et les ordonner dans des structures de données. Finalement, nous avons tenté de les regrouper en nous basant sur ces règles pour créer des hiérarchies triangulaires et hexagonales selon l'approche bottom-up. Les sections suivantes abordent chaque étape de cette démarche.

4.2.1 Partition de l'espace

Dans cette section nous présentons la première idée à laquelle nous avons eu recours pour créer plusieurs triangles équilatéraux dans le but de découper l'espace. Dans une première étape nous chercherons à construire une fractale triangulaire à partir d'un grand triangle initial. Dans une seconde étape, nous chercherons à étaler ce motif sur l'ensemble de notre espace. Les sections qui suivent illustrent cette démarche.

4.2.1.1 Partition d'un triangle

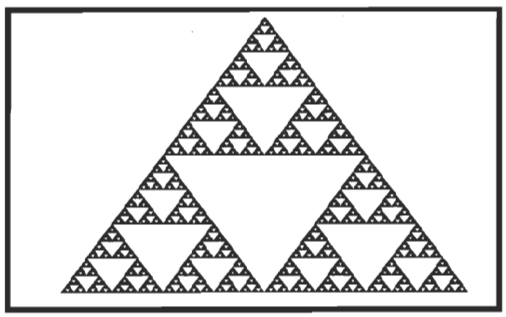


Figure 4.1 Triangle de Sierpenski (Rousseau, 2007)

Nous avons considéré l'algorithme de Sierpinski qui produit une fractale à base de triangles, comme le montre la figure précédente. Toutefois, cet algorithme n'est pas adapté pour fournir un triangle entier à cause du creux au niveau des triangles des milieux. Il a fallu donc le modifier pour couvrir tout l'espace à l'intérieur de ce genre de triangles. En d'autres termes, il faut remplir les creux par d'autres triangles plus petits. L'algorithme est le suivant :

```
SierpinskiTriangle (int[] x, int[] y, int d)
Debut
        Si (d < 20) Alors
            Créer point A (x[0], y[0])  et Créer point B (x[1], y[1])
            Créer point C (x[2], y[2])  et  Créer Triangle(A,B,C)
            Ajouter triangle à ListeTri
         Sinon
           // points milieux des côtés du triangle
            int xMc = (x[0] + x[1]) / 2, yMc = (y[0] + y[1]) / 2
            int xMb = (x[0] + x[2]) / 2, yMb = (y[0] + y[2]) / 2
            int xMa = (x[1] + x[2]) / 2, yMa = (y[1] + y[2]) / 2
            int[] xNouveau1 = { x[0], xMc, xMb }
            int[] yNouveau1 = { y[0], yMc, yMb }
             SierpinskiTriangle (xNouveau1, yNouveau1, d / 2)
            // points haut des côtés du triangle
             int[] xNouveau4 = { xMa, xMb, xMc }
            int[] yNouveau4 = { yMa, yMb, yMc }
            SierpinskiTriangle (xNouveau4, yNouveau4, d / 2)
           // points gauches des côtés du triangle
            int[] xNouveau2 = { x[1], xMc, xMa }
            int[] yNouveau2 = { y[1], yMc, yMa }
            SierpinskiTriangle(xNouveau2, yNouveau2, d / 2)
            // points droits  des cotés du triangle
            int[] xNouveau3 = { x[2], xMb, xMa }
            int[] yNouveau3 = { y[2], yMb, yMa }
            SierpinskiTriangle ( xNouveau3, yNouveau3, d / 2)
        FinSinon
```

Figure 4.2 Algorithme de Sierpenski modifié

Cet algorithme admet comme paramètres d'entrée: la valeur du côté d'un triangle (*d*) et deux listes qui enregistrent les valeurs en x et en y de chaque segment d'un triangle. Le cas trivial est quand *d* est inférieur à une valeur par défaut (20 dans notre exemple). Il est à remarquer que nous avons choisi la valeur 20 pour tester notre algorithme, mais par la suite cette distance sera paramétrable par l'usager.

Le cas récursif est un simple calcul mathématique. En effet, à partir des listes qui sont passées comme paramètres, on calcule les coordonnées du point qui représente le milieu de chaque segment d'un triangle. Ces points représenteront, à chaque itération, les sous-triangles d'une taille égale à d/2 (la moitié de la valeur initiale (*d*). Aussi, ces points seront sauvegardés dans des nouvelles listes qui serviront comme paramètres d'entrée au même algorithme, à la suite d'un appel récursif. Il est à remarquer que nous avons utilisé une liste comme structure de données pour enregistrer les points d'un seul triangle. Ainsi, chaque

triangle n'est qu'une liste contenant les coordonnées de ses trois sommets. Pour conclure, dans l'algorithme originel, le partitionnement est seulement effectué pour les trois sommets d'un triangle qui se situent en haut, à gauche et à droite de ce dernier. Notre modification réside dans l'ajout et le calcul de nouveaux points qui se trouvent au milieu des segments d'un triangle. Par conséquent, au milieu de chaque grand triangle, des sous-triangles nouvellement construits remplissent les creux du triangle de Sierpenski. Dans ce qui suit, nous appelons 'triangle de Sierpenski modifié' le résultat de notre algorithme.

4.2.1.2 Partition de l'espace par des triangles

Comme nous l'avons mentionné dans la section précédente, afin de couvrir tout l'espace, il faut étaler le triangle de Sierpinsky modifié sur l'ensemble de notre espace et dans les deux sens. On réfère au sens d'un triangle par la direction vers laquelle pointe son sommet d'en haut. Il peut pointer vers le haut ou vers le bas. À cette fin, il faut imbriquer l'algorithme précédent dans une boucle de répétition et commencer à partir d'un point (x0, y0) dont les coordonnées sont confondues avec le point de l'origine de l'écran de coordonnées (0,0). Ce point représentera le point à partir duquel on va dessiner chaque nouveau triangle. Rappelons qu'en dessin 2D, ce point est le plus haut à gauche et que l'axe des y se situe à gauche et qu'il est orienté de haut vers le bas. Quant à l'axe des x, il se trouve en haut et il est orienté de gauche à droite. L'intersection de ces deux axes est le point à l'origine. Dans ce qui suit, nous expliquons cet algorithme. Ce dernier admet comme paramètres d'entrée la largeur et la longueur d'un espace à partitionner. Il commence par initialiser les listes des coordonnées des points d'un triangle en ajoutant au point de coordonnées (x0, y0) la hauteur h et la valeur de l'arrêt d du triangle. Une fois que cette initialisation est terminée, on appelle la fonction Sierpenski, que nous avons présentée à la section précédente, pour dessiner un seul triangle de Sierpenski modifié, qui est orienté vers le haut. Lors de l'itération suivante, le prochain triangle de Sierpenski modifié doit être construit selon l'axe des X et il doit être orienté vers le bas. Pour ce faire, il faut considérer le triangle précédent, ajouter la valeur d'un arête à ses sommets et considérer les points qui leurs sont symétriques par rapport à l'axe Y. Bref, pour deux itérations consécutives, il faut avoir un triangle de Sierpenski modifié orienté vers le haut, suivi d'un autre triangle orienté vers le bas. Finalement, il faut déplacer le premier point x0 de sorte à ce qu'il soit confondu avec l'extrême point gauche du dernier triangle fraîchement créé. C'est avec cette méthode

que nous pouvons dessiner un triangle selon l'axe des X. Une fois que la largeur est atteinte, on se déplace verticalement en affectant à *x0* la valeur *0* et en augmentant *y0* de *h*. L'output de cet algorithme est un ensemble de triangles couvrant un espace et qui seront enregistrées dans une liste. L'algorithme est illustré par la figure 4.3.

```
PartitionnerEspace(largeur, longeur)
Initialiser h la hauteur d'un triangle
Répéter
    Pour  j = 0; j < largeur; j++
     Début
            xA = x0;
            yA = y0 + h;    // (bas-gauche)
             xB = x0 + d;
             yB = y0 + h;    // (bas-droite)
             xC = x0 + d / 2;
             yC = y0;
           int[] x = { xA, xB, xC };
           int[] y = { yA, yB, yC };
           int[] xx = { xC, xC + d, xB };        // inverser les coordonnées sur l'abscisse
           int[] yy = { yC, yC, yB };            // garder les mêmes coordonnées
           SierpinskiTriangle(x, y, d / 2);
           SierpinskiTriangle(, xx, yy, d / 2);
           x0 = xC + d / 2;                              // déplacer x0
        Fin Pour
     x0 = 0;
     y0 = h + (i * h);
  Jusqu'à (i++ <  longueur)
```

Figure 4.3 Algorithme de partitionnement de l'espace

Du fait que nous travaillons sur un espace très grand, qui dépasse les limites de l'écran de l'ordinateur, les techniques du « pan » et du « zoom » présentées dans le chapitre 2 sont primordiales. A cette fin, nous avons choisi d'utiliser la librairie Piccolo.Net. Rappelons que notre premier défi est de partitionner l'espace avec des triangles. En d'autres termes, il faut dessiner sur le canvas 2D de notre application des triangles d'une certaine taille qui peut être paramétrable.

4.2.1.3 Résultat et conclusion

La figure suivante montre une portion de la grille après une décomposition selon l'algorithme de Sierpinski que nous avons adapté.

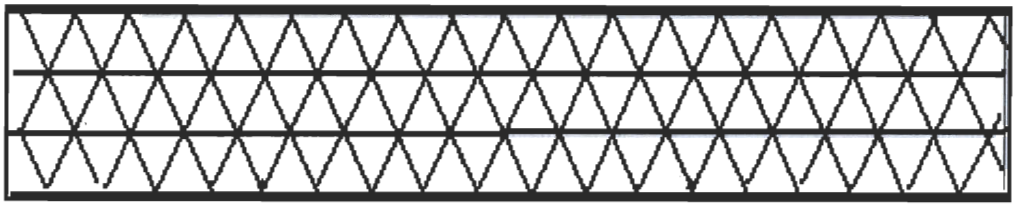


Figure 4.4 Grille triangulaire créée avec l'algorithme de Sierpinski

Sur un plan visuel, les triangles constituent une grille triangulaire, mais ils n'ont aucune relation spatiale ou hiérarchique entre eux. Par conséquent, il faut procéder à un indexage pour ordonnancer les cellules, et les regrouper pour former des hexagones. Nous avons exploré dans un premier temps, une approche ascendante « bottom-up » pour réaliser ces hiérarchies triangulaires indexées. Nous présenterons la démarche ainsi que ses limites. Finalement nous présenterons le résultat obtenu et nous concluons.

4.2.2 Essai d'une approche Bottom -up

L'objectif de l'approche « bottom-up » est de regrouper des cellules de petites tailles, en des cellules de plus grandes tailles qui se trouvent à un niveau supérieur. En d'autres termes, les cellules d'un niveau i, et de taille t seront englobées par les cellules du niveau $(i+1)$ et de tailles $(t * f)$ ou f est appelé le facteur de résolution. Ceci est conforme à l'approche de Tsui (Tsui, 2002). Rappelons aussi que Paris (Paris, 2007) utilise la même approche pour regrouper des triangles irréguliers construits à partir d'une triangulation de Delaunay. En s'inspirant de cette approche, les cellules inférieures seront regroupées dans des cellules de niveaux supérieurs et de plus grandes tailles. Cette abstraction est construite en deux étapes : Premièrement, il faut trouver les n voisins immédiats d'une cellule donnée. Deuxièmement, il faut regrouper les cellules de ce voisinage jusqu'à ce qu'il n'en reste qu'une seule, qui appartiendra au niveau le plus haut et dont la taille est maximal. Pour ce faire, nous avons essayé deux stratégies d'indexage que nous présenterons dans les sections suivantes.. La première stratégie est basée sur des règles géométriques, tandis que la deuxième utilise des règles d'indexage.

4.2.2.1 Règles Géométriques

Nous présentons dans ce qui suit la définition et l'intérêt des règles géométriques ainsi que le formalisme que nous avons choisi pour les caractériser.

4.2.2.1.1 Utilité et définition des règles

Les règles géométriques représentent un mécanisme de comparaison spatiale entre les triangles en fonction des directions (droite, gauche, verticale). Elles considèrent également le sens d'orientation vers lequel pointe le sommet d'un triangle (haut, bas). Pour chaque sens d'orientation et pour chaque direction, nous avons établi une règle. Grâce à ces règles on peut :

- Trouver le voisinage d'une cellule triangulaire.
- Comparer deux triangles selon un critère spatial (direction ou orientation).
- Établir des relations d'adjacence entre deux triangles.
- Déterminer l'ensemble des configurations spatiales possibles entre deux triangles.

Nous appelons configuration spatiale la position d'un triangle par rapport à un autre. Cette configuration est géo-référencé dans un espace 2D qui a comme repère les axes X et Y de l'écran. Rappelons que cette notion est abordée par Samet (Samet, 1999). Elle a énuméré dans une table d'états binaires de l'ensemble des configurations de son approche. Rappelons également que Fekete (Fekete,1993) a listé l'ensemble des cas possibles entre chaque sens et chaque direction et il leur a affecté des numéros. C'est par le biais de ces numéros qu'il caractérise sa configuration spatiale et par le fait même il localise deux triangles dans l'espace. Dans ce qui suit nous allons présenter le formalisme que nous avons choisi pour décrire une règle sous une forme générique.

4.2.2.1.2 Formalisme des Règles

En fait, pour chaque genre de configuration entre deux triangles, ou encore pour chaque relation spatiale entre deux triangles, nous avons défini une règle globale R. Cette dernière est composée par des sous-règles élémentaires qui comparent les sommets ou bien les côtés. À ces sous-règles, on applique le « et » logique. Seulement, la valeur « vraie » du ET permet de satisfaire la règle globale et par le fait même, de valider la configuration spatiale stipulée par cette dernière. Nous avons choisi le formalisme suivant pour présenter les règles :

- **Nom :** Définit le nom d'une configuration spatiale possible entre deux triangles A et B. A représente toujours la cellule triangulaire d'intérêt. B représente la cellule à laquelle on applique la règle pour déterminer son adjacence avec A.
- **Objectif :** Décrit la configuration spatiale d'une règle.

- **Ri :** Identifie la règle numéro i qui s'applique au triangle B.
- **Sous-règles *Ri.j*** : Détermine les sous-règles numéros j de la règle *Ri*.
- **Complétion :** La manière dont les règles élémentaires sont utilisées pour satisfaire *Ri*.

Comme nous l'avons mentionné dans la section 4.2.2.1.1, ces règles servent à établir une relation d'adjacence entre deux triangles. Dans notre contexte, il existe deux types d'adjacence. La première est une adjacence par le côté, c'est-à-dire si les deux triangles ont une arête en commun. La deuxième est l'adjacence par sommet, dans le cas où les deux triangles ont un point en commun. Ceci implique qu'il y a des règles spécifiques à chaque type d'adjacence. Dans ce qui suit nous présentons ces règles selon le type d'adjacence montrant ainsi les différentes configurations spatiales possibles.

4.2.2.2 Règles d'adjacence triangulaires

La section 4.2.2.2.1 présente le premier type d'adjacence qui est l'adjacence par sommet. Quant à la section 4.2.2.2.2, elle présente le deuxième type, qui est l'adjacence par côté. Dans chaque section, les tableaux montrent les configurations spatiales possibles entre deux triangles A et B qui sont orientés soit vers le haut, soit vers le bas. Par ailleurs, les points noirs sur ces schémas montrent les sommets impliqués dans le premier cas d'adjacence. Finalement, ils présentent aussi comment on applique le formalisme de règles cité dans la section 4.2.2.1.2.

4.2.2.2.1 Adjacence par Sommet :

Si deux triangles ont un point en commun, on dit qu'ils sont adjacents par sommet. Ce cas ne surgit que si les deux triangles appartiennent au même grand triangle englobant, c'est-à-dire le triangle dont le côté est égal à la somme des deux arêtes des deux sous-triangles.

a) Adjacences par sommet des triangles orientés vers le haut :

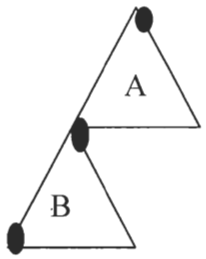
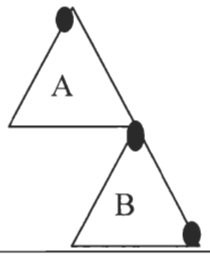
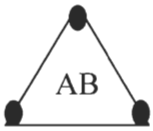
R1 : Nom : Triangle B Inférieur à Gauche **Objectif :** précise si un triangle est inférieur sur la gauche d'un autre triangle. **Sous-Règles :** **R1.1** : le point gauche de A et le sommet haut de B sont confondus **R1.2** : le point gauche de A est plus haut que le point gauche de B **Complétion :** *Si R2.1 et R2.2 alors B est inférieur à gauche de A*	
R2 - Nom : Triangle B Inférieur à Droite **Objectif :** Précise si un triangle est inférieur à la droite d'un autre triangle. **Sous-Règles :** **R2.1** : le point droit de A et le sommet haut de B sont confondus **R2.2** : le point droit de A est plus haut que le point droit de B **Complétion :** *Si R1.1 et R1.2 alors B est inférieur à Droite de A*	
R3 : Nom : Triangle B confondu à A **Objectif** : Précise si deux triangles sont identiques. **Sous-Règles :** **R3.1** le point gauche de A et le point gauche de B sont confondus **R3.2** le point droit de A et le point droit de B sont confondus **R3.3** le point haut de A est confondus au point haut de B. **Complétion :** ***Si*** *R3.1 et R3.2 et R3.3 alors B est confondu à A*	

Tableau 4-1 Règles d'adjacences par sommet des triangles orientés vers le haut

b) Règles d'adjacences par sommet des triangles orientés vers le bas :

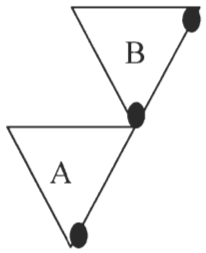
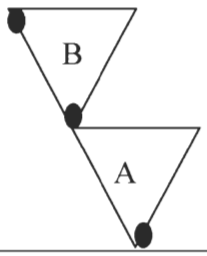
R4 - Nom : Triangle B Supérieur à Droite **Objectif :** Précise si un triangle B est supérieur à droite d' un autre triangle A. **Sous-Règles :** **R4.1** le point droit de A et le sommet de B orienté vers le bas sont confondus **R4.2** le point droit de A est inférieur au point droit du B **Complétion :** *Si R4.1 et R4.2 alors B est Supérieur à Droite de A*	
R5 : Nom : Triangle B Supérieur à Gauche **Objectif :** Précise si un triangle B est supérieur sur la gauche d' un autre triangle A. **Sous-Règles :** **R5.1** le point gauche de A et le sommet de B sont confondus **R5.2** le point gauche de A est plus bas que le point gauche du B **Complétion :** Si R5.1 et R5.2 alors B est supérieur à gauche de A	

Tableau 4-2 Règles d'adjacences par sommet des triangles orientés vers le bas

4.2.2.2.2 Règles d'adjacence par côté

Les règles précédentes définissent les voisins immédiats appartenant au même parent, c'est-à-dire toujours au sein du même triangle englobant. Cependant, qu'arrive t-il si le cas inverse surgit, c'est-à-dire, les deux triangles n'appartiennent pas au même triangle englobant ? En fait, ce cas permettra de naviguer entre deux triangles englobants. À titre d'exemple, en appliquant l'algorithme de l'ancêtre commun (Samet, 1999) aux petites cellules, on peut transiter d'un triangle englobant vers un autre comme le montrent les figures suivantes :

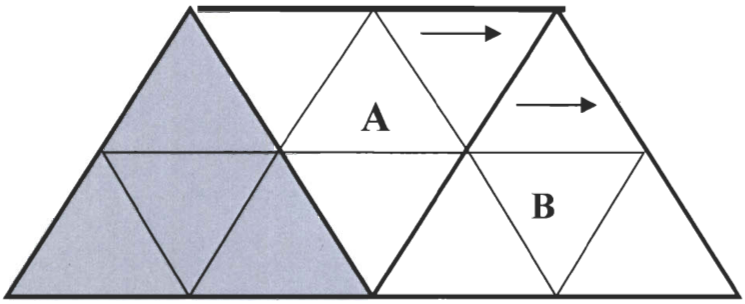


Figure 4.5 Transitions de gauche à droite entre deux triangles englobant

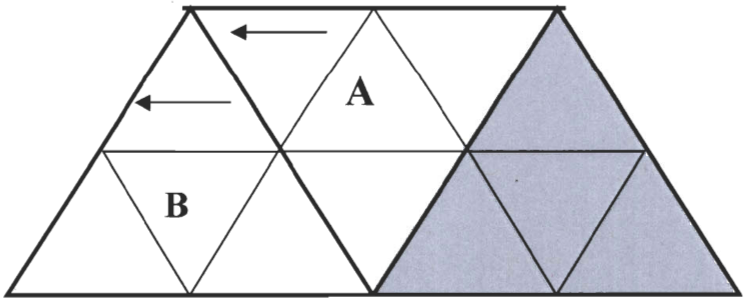
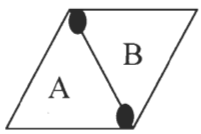


Figure 4.6 Transitions de droite à gauche entre deux triangles englobant

Samet (Samet, 1999) a utilisé le même principe en appliquant des décalages aux bits du « paths array ». En ce qui nous concerne, nous avons résolu ceci en appliquant des règles d'adjacence par le côté afin de déterminer si les deux triangles appartiennent au même niveau et à deux parents englobant différents. Nous avons utilisé des règles d'indexage au lieu des décalages des bits pour implémenter ce procédé. Nous allons aborder ceci dans la section des règles d'indexage. Le tableau suivant indique les règles d'adjacence par coté que nous avons utilisés.

R7 : Nom : Triangle B Adjacent à Droite **Objectif :** précise si un triangle est adjacent par coté en commun. **Sous-Règles :** R7.1 le point haut de A et le point gauche de B sont confondus. R87.2 le point droit de A et le point haut de B sont confondus. **Complétion :** Si R7.1 et R7.2 alors B est adjacent à droite de A	

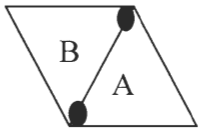
R11 : Nom : Triangle B Adjacent à Gauche **Objectif :** précise si un triangle est adjacent par côté droit commun. **Sous-Règles :** R11.1 le point haut de A et le point droit de B sont confondus. R11.2 le point gauche de A et le point haut de B sont confondus. **Complétion :** Si R11.1 et R11.2 alors B est adjacent à gauche de A	

Tableau 4-3 Ajacences par côté des triangles

Après avoir présenté l'ensemble de ces règles, nous allons examiner la manière dont elles sont utilisées pour atteindre nos objectifs. La section 4.2.2.3 présente les algorithmes que nous avons trouvés en utilisant les règles géométriques.

4.2.2.3 Algorithmes et utilisation des règles

Une fois que ces règles sont fixées, on les utilise pour comparer la position d'un triangle B par rapport à celle du triangle d'intérêt A. Ceci implique que pour chaque triangle de la grille, il faut déterminer sa position spatiale en vérifiant toutes les règles présentées plus haut et ce, par rapport à *n* autres triangles dans un voisinage donné. Une fois cette relation trouvée, on construit une cellule triangulaire de plus grande taille en cherchant les points extrêmes de chaque sous-triangle et créant de la sorte un nouveau triangle plus grand. En d'autres termes, ce dernier est le triangle parent qui sera créé à partir des points extrêmes de droite, de gauche, et de haut des trois triangles fils. C'est ainsi que le regroupement géométrique fonctionne dans le cas de l'approche bottom-up.

Cet algorithme est formé par deux boucles répétitives. L'idée est de comparer chaque triangle par rapport aux autres selon les règles précédentes. En d'autres mots, pour chaque triangle de la grille, il faut trouver son voisin d'en haut, de gauche et de droite. Ensuite, il faut chercher le point haut du triangle d'en haut, gauche du triangle de gauche et droite du triangle de droite pour trouver les sommets du triangle parent. Finalement, on ajoute les triangles fils à la liste des triangles des parents. Le triangle de milieu n'est pas inclus dans ce processus, mais il est ajouté aussi à cette liste puisqu'il est considéré comme un enfant du triangle parent trouvé. Pour savoir qu'il s'agit d'un triangle du milieu, une fois que le

triangle de gauche et de droite sont trouvés, il suffit d'appliquer de nouveau la règle d'adjacence par côté pour trouver le triangle du milieu. Ce dernier doit se trouver entre les celui de gauche et celui de droite. Une liste de numéros est initialisée par défaut afin de définir un type de configuration spatiale. En appliquant une règle, un type de retour qui appartient à cette liste est vérifié à chaque fois. La valeur -1 est retournée le cas échéant. L'algorithme est le suivant :

```
Regrouper()
Pos =0
Intialiser les listes a vide
Pour chaque Triangle j dans la liste List
 Triangle  A  =  List ( j )
  Pour chaque Triangle i dans la liste List et i différent de j
        Triangle  B =  List ( i )
        Entier valeur = AppliquerRegle ( A , B )
              Si valeur appartient à l'ensemble des valeurs attendus I
                    Ajouter B à la position pos dans la  Liste LV des voisins de A et pos ++
    Fin Pour
Tantque  j est inférieur à pos
        ListSommet = Obtenir les Sommet des Triangles voisins de A dans LV
        TriangleParent = Crée Triangle(ListSommet)
FinTantque
 Ajouter TriangleParent à la lise des Parent de la cellule A.
Fin Pour
```

Figure 4.7 Algorithme de regroupement selon l'approche bottom up

Cet algorithme est de l'ordre de O (n^2). Ce qui implique une mauvaise performance au niveau de la hiérarchisation de la grille lors de l'exécution. En effet, pour chaque cellule, il faut exécuter cet algorithme. Pour un jeu de données de quelques milliers de cellules, ceci est réaliste, mais pour des millions de cellules ceci est inconcevable. Vu notre contexte d'application où l'on risque d'utiliser un très grand nombre de données et des espaces volumineux, cette solution est écartée.

À l'image des règles géométriques, nous avons eu l'idée d'utiliser un autre type de règles : les règles d'indexation. Nous présentons ce genre de règles dans la section suivante.

4.2.3 Règles d'indexation

Dans cette section, nous allons examiner les règles d'indexation. Nous allons présenter leur principe ainsi que leur intérêt. Ensuite, nous allons présenter la liste des règles que nous avons trouvée. Finalement, on présente les résultats obtenus.

4.2.3.1 Principe

À l'image du travail de Samet (Samet.1999) qui a utilisé des opérations mathématiques sur les séquences binaires, nous avons eu l'idée d'appliquer une approche équivalente en considérant les index des cellules dans la liste globale comme des identifiants uniques de chaque cellule. Notons que la liste globale est un tableau contenant tous les triangles issus de la partition de l'espace (voir section 4.2). Par conséquent, pour trouver les voisins ou les parents, on applique des opérations d'additions, de soustractions et de multiplications aux index de la liste globale.

4.2.3.2 Intérêt des règles d'indexation

Les règles d'indexation servent à trouver le voisinage d'une cellule donnée. Rappelons, qu'un voisinage triangulaire est l'ensemble des cellules adjacentes à une cellule donnée et aussi adjacentes entre elles. L'ensemble de cette agglomération de petites cellules peut former une cellule triangulaire englobante de dimension plus grande. Ainsi, pour trouver ce voisinage, une cellule doit trouver ses voisins et les voisins des voisins, qui se situent à droite et à gauche, de cette dernière. Par conséquent, notre mécanisme de règles doit assurer la transition d'une cellule à une autre horizontalement. C'est ce que Samet (Samet,1999) qualifie par les transitions horizontales. Ce ne sont que des transitions de gauche à droite et vice-versa entre deux cellules. Contrairement à cela, il existe aussi les transitions verticales (Samet,1999). C'est le fait de trouver les cellules voisines en haut ou en bas d'une cellule triangulaire donnée.

Les règles d'indexation permettront aussi d'extraire les sommets des triangles enfants pour former un nouveau triangle parent, plus grand afin de créer la hiérarchie. Ainsi, le triangle fraîchement créé aura comme descendants l'ensemble des triangles qui formaient ce voisinage, comme l'illustrent les figures 4.8 et 4.9.

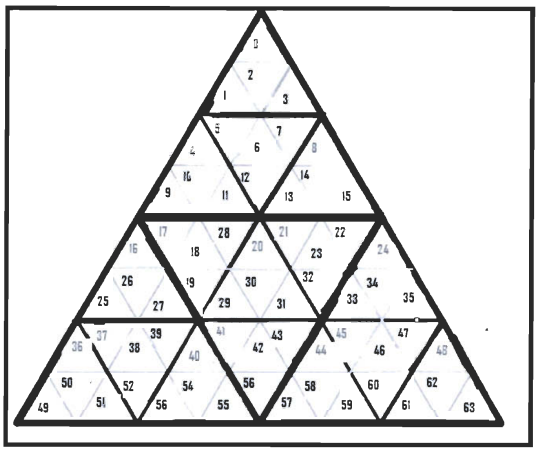


Figure 4.8 Index des triangles Enfants dans un triangle de taille 8

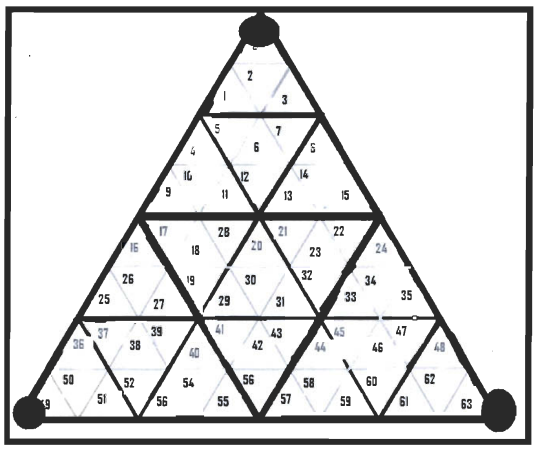


Figure 4.9 Regroupement et extraction des sommets

La première figure montre les index des sous-triangles obtenus grâce aux règles d'indexation. La 2^{ième} figure montre les sommets du plus grand parent qu'on peut extraire en appliquant ces règles à chaque niveau et par le fait même agréger les sous-triangles en des triangles de plus haut niveau.

4.2.3.3 Liste des règles d'indexage

Nous présentons la liste des règles que nous avons trouvées selon une approche essai/erreur. Dans tout ce qui suit, on considère que *n* est la taille d'un triangle. La première partie présente les règles qui régissent les transitions horizontales. La partie suivante présente les transitions verticales.

i) Transitions horizontales :

- L'index du sommet Gauche se trouve toujours à gauche en partant du premier triangle rencontré.
- Les index des sommets de gauches sont trouvés comme suit : *(n + 2)* .
- Il faut que n soit augmentée de 2 pour avoir toujours 4 enfants.
- Pour i triangles orientés vers le haut, il y a *(i – 1)* triangles orientés vers le bas.
- L'index du sommet du triangle orienté vers le bas = l'index du sommet du triangle orienté vers le haut + 1
- L'index du prochain somment de triangle orienté vers haut = l'index du dernier sommet orienté vers haut + 1

ii) Transitions verticales :

Dans le cas d'un déplacement vertical, les règles dépendent de l'orientation du sommet de la cellule d'intérêt. Nous distinguons 2 cas :

- ***Cas 1 : Sommet du triangle orienté vers le haut***
  - L'index de l'enfant de gauche = l'index du sommet de gauche + *pas*, le *pas* est incrémenté de 4 à chaque nouvel étage et initialement égale à 3.
  - L'index de l'enfant vertical = l'enfant de gauche + 1.
  - L'index de l'enfant droit = l'enfant vertical + 1.
- ***Cas 2 : Sommet du triangle orienté vers le bas***
  - L'index de l'enfant de gauche = = l'index du sommet du triangle orienté vers le bas +1
  - L'index de l'enfant vertical = = l'index de l'enfant de gauche + 1.
  - L'index de l'enfant droit = l'index du sommet + pas, le pas est un nombre impair égal à 3 est augmenté de 4 à chaque étage.

4.2.3.4 Conclusion et résultats

À cause de notre approche de décomposition de l'espace à l'aide de l'algorithme de Sierpinski modifié, que nous avons présenté dans la section 4.1, la distribution des cellules dans la liste n'est pas uniforme, puisque la courbe d'indexation de l'espace est une courbe de Sierpinski (voir chapitre 3). Par conséquent, un élément voisin à une cellule donnée

n'est pas forcément celui qui se trouve à l'index suivant. Nous avons besoin alors d'une partition selon un « row-order » ou en « column-order », où les triangles sont l'un à la suite de l'autre ainsi que leurs index dans le but de faciliter les opérations d'indexage. Malgré ces contraintes, nous avons réussi à obtenir une cellule hexagonale englobant d'autres cellules triangulaires. La figure 4.10 montre la cellule résultante. Sur cette figure, chaque couleur représente un niveau hiérarchique. On distingue 4 couleurs et par conséquent 4 niveaux.

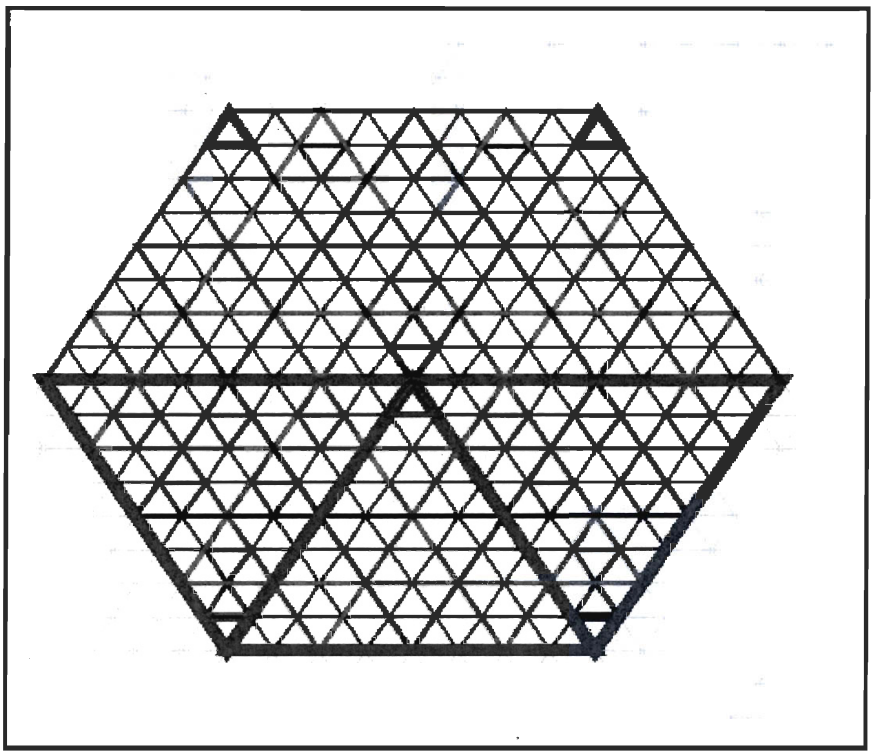


Figure 4.10 Cellule hexagonale issue de l'essai non concluant

Tout ce processus a servi pour construire une seule cellule qui est présentée par la figure précédente. Ce procédé était peu performent et gourmand en termes de temps. Du fait qu'une grille est formée par des milliers, voire des millions de cellules pareilles nous avons écarté cette approche. Toutefois, nous nous sommes basés sur notre expérience lors de cet essai pour dégager quelques besoins que nous avons jugé pertinents pour créer une structure générique. Ces besoins sont :

- Avoir un facteur de résolution paramétrable selon le contexte d'utilisation. C'est-à-dire, théoriquement, une cellule peut se regrouper *n* fois. La seule limite possible est le compromis entre ce facteur de composition et les performances souhaitées.
- Pouvoir modifier la taille de la cellule pour créer un mécanisme qui s'apprête mieux au « zooming ». ---
- Tirer profit de la récursivité afin d'avoir une solution rapide et naturelle au contexte de hiérarchie.
- Éviter d'appliquer des opérations de comparaisons pour trouver la relation enfant/ parent

À partir de ces hypothèses, nous avons conclu que l'approche bottom-up est inadéquate parce qu'elle est :

- Appropriée dans un contexte ou les voisins sont connus, comme dans les travaux de Paris (Paris, 2007) : les voisins des cellules irrégulières sont définis par les arcs des graphes de voisinages. Dans notre cas, comme les cellules sont régulières, il faut chercher les voisins d'une cellule donnée. Donc une complexité de O(n) au meilleur cas.
- Les règles d'indexages ne sont pas constantes. Si on change les paramètres d'entrée, la répartition des éléments dans la liste va changer, par conséquent il faut recommencer tout le processus de nouveau.
- Le partitionnement de l'espace est inapproprié et par conséquent la manière de découper l'espace ne permet pas un mécanisme d'indexage facile.

C'est pour ces raisons que nous avons adopté une autre approche, opposée èa celle-ci, soit l'approche « top-down ». Elle a été explorée dans le cas d'un autre essai, qui a été concluant et que nous allons aborder dans la section 4.3.

4.3 Essai Concluant

Le premier essai a servi de premier prototype afin de découvrir les contraintes techniques et théoriques relatives à notre problématique. Dans ce qui suit, nous proposons la démarche finale pour solutionner notre problème.

4.3.1 L'approche top down

Contrairement à l'approche précédente, qui était ascendante, cette approche est descendante et dite « top down ». En fait, partant d'une cellule de plus haut niveau et de plus grande taille, on la décompose en des cellules de plus bas niveau et de plus petites tailles. Le partitionnement se fait en tirant profit de la récursivité. En effet, chaque cellule triangulaire de grande taille d'un niveau supérieur se divise en 4 cellules enfants de plus petites tailles, appartenant à un niveau inférieur, exactement à l'image des travaux des quadtrees rectangulaires (Samet 1999) et (Schrack, 1992). Ainsi, on pourrait avoir plusieurs niveaux de hiérarchies une fois que la relation enfant-parent est établie. Nous visons à ce que cela soit atteint à l'aide d'algorithmes récursifs. Nous visons à fixer la taille d'une cellule ainsi que le niveau de résolution, tout comme Tsui (Tsui et al, 2000) le fait dans son framework. Effectivement, tout d'abord, ce dernier fixe la taille de son unité atomique de base « BUT», ensuite il précise le nombre de décompositions qu'il veut obtenir par cellule, ou encore le nombre de niveaux hiérarchiques à atteindre. Finalement, il commence son partitionnement.

Dans notre cas, le nombre total des cellules est géré à partir des dimensions de la grille qu'on cherche à construire. En fixant la longueur et la largeur de la grille, on peut obtenir le nombre désiré. Ainsi, on configure notre partitionnement selon :

- *La taille d'une cellule*
- *Le nombre de niveaux*
- Le nombre total des cellules à obtenir

4.3.1.1 Avantages de cette approche

L'avantage majeur de cette approche est le fait d'obtenir une solution générique paramétrable qui tient compte de la taille, du nombre des éléments et des dimensions de la grille afin de créer des cellules hexagonales hiérarchiques. Un autre avantage de cette approche, est que les n voisins immédiats sont calculés au fur et à mesure de la décomposition, il suffit juste de les insérer dans la structure et la relation composée/composite est satisfaite de manière naturelle grâce à la récursivité. Par contraste dans l'approche précédente, une fois que nous avions trouvé les *n* voisins immédiats, il fallait calculer les parents et établir la relation parent\enfant et finalement les insérer dans la

structure. Par conséquent, l'approche « top-down » est plus souple et rapide que l'approche « bottom-up ».

4.3.1.2 Inconvénients de cette approche

L'inconvénient de l'approche descendante, est sa limite en termes de niveaux d'abstraction ou de niveaux de hiérarchie. En effet, comme le mentionne Samet (Samet, 1999), il n'y a pas une corrélation entre les lignes et les colonnes formées par les cellules triangulaires. Donc on ne peut pas appliquer les mécanismes d'indexage traditionnels comme dans les grilles rectangulaires.

4.3.1.3 Conclusion

Selon le principe de cette approche, nous allons essayer de partitionner l'espace et nous tenterons de répondre aux objectifs que nous avons mentionnés dans l'introduction. Dans ce qui suit, on présentera notre démarche et notre solution.

4.3.2 Partition de l'environnement

Cette section présente notre solution et nos algorithmes pour partitionner un espace selon la philosophie de l'approche « top down ».

4.3.2.1 Partition de l'espace

La première étape consiste à obtenir des cellules triangulaires d'une grande taille *d*, couvrant l'ensemble de notre espace de travail. Pour simplifier ce défi, avant toutes

```
PartitionnerTriangle( d )
Point Obas = (0,0)          Initialiser le point d'origine  d un triangle orienté vers le bas
Point Ohaut = (0,0)         Initialiser le point d'origine  d un triangle orienté vers le haut
Point D =  (0,0)            Initialiser le point de droite
Point G = (0,0)             Initialiser le point de gauche
Point H =  (0,0)            Initialiser le point de haut
Côté x = valeur             Initialiser le côté d'un triangle
Hauteur = valeur            Initialiser la hauteur  de la grille
Longueur = valeur           Initialiser la longeur de la grille
 Pour l = 0, l <= hauteur,  l++
    l =l + l
        Dessiner le premier triangle pointant vers le bas
        Dx  = O.x + ( d * x ), Dy =  Oy
        Hx = Ox + (d/2*x),    Hy=  Oy + (d/2 * x)
        Ohaut = H
        Tab1 [] = G, Droite, H; t1 = tab1
        Ajouter t1 à la structure   et Dessiner le triangle t1 et t1=null
       Dessine les triangles sur les colonnes
        Pour i = h; i  <  hauteur; i++
              G= H, H = D
               Dx = Gx + (1 * (d*x) ), Dy = Gy
               Tab1 [] = G, D, H
                t1 = tab1
               Ajouter t1 à la structure et Dessiner le triangle t1
                t1 = null
         FinPour

        H = Ohaut Dessiner le premier triangle pointant vers le haut
        Gx = Ohaut - (d/2*x), Gy= ( (d*x) * l) )
        Dx = Gx + (1 * (d * x)), Dy = Gy
        tab2 [ ]= G, D, H
        Triangle t2 = tab2
        Ajouter t2 à la structure  et Dessiner le triangle t2
         t2 = null ; Obas = G
        Dessine les triangles sur des lignes
         Pour  k = 1;  k <   hauteur;   k++
                G = H; H = D
                Dx = Gx + ( d *x ) , Dy = Gy
                 tab3 [ ] = G, D, H
                 Triangle t3 = tab3
               Ajouter t3 à la structure  et Dessiner le triangle t3 et t3 = null;
        FinPour
        Gx = 0,  Gy =  1 * ( d*x )
  FinPour
```

Figure 4.11 Algorithme de dessin des triangles

L'algorithme ressemble aux algorithmes basés sur les fractales. Il admet le paramètre *d* qui représente la taille d'une cellule triangulaire. Nous appelons point *O*, un point dans l'espace, qui caractérise un triangle donné à un instant donné lors du processus de construction de ce triangle. C'est à partir des coordonnées de ce point que nous allons dériver celles des autres points du triangle en cours de construction. L'algorithme fixe le point *O* à l'origine de l'écran, deux autres points dans l'espace, et il trace des droites entre eux de sorte que les trois forment un triangle équilatéral. Ces droites ont la même longueur. Ensuite, le dernier point trouvé, c'est-à-dire celui au sommet d'un triangle orienté vers le bas, devient le point à l'origine. En fait, on calcule le point de droite à partir de celui qui est à l'origine. Ce dernier devient le point de gauche, à partir duquel on déduit celui du sommet. Finalement, une boucle répétitive est utilisée pour étaler ce triangle tout le long d'une colonne. Dans ce cas, les triangles sont symétriques. Il existe deux cas de symétrie comme le montre la figure 4.12.

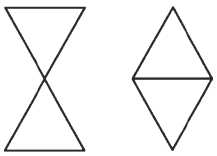


Figure 4.12 Les deux cas de symétrie

Dans le premier cas, nous affectons le dernier point trouvé dans un triangle qui pointe vers le bas au nouveau point du sommet d'un triangle qui pointera vers le haut. Dans l'autre cas, on affecte soit le point de gauche, soit celui de droite au point d'origine, qui sera respectivement celui de droite ou celui de gauche du nouveau triangle qui s'orientera vers le bas. Le premier triangle de la première colonne étant formé, on extrait son point de gauche, et on l'affecte au point à l'origine. Ainsi on fabrique le 1^{er} triangle de la $2^{ième}$ colonne et on réitère jusqu'à ce qu'on construise un nombre de triangles égal à la hauteur h, qui est déjà fixée. D'ailleurs, c'est l'intérêt de la $2^{ième}$ boucle *Pour* dans notre algorithme. Enfin, ce pattern est répété autant de fois que la largeur L. Enfin, chaque triangle créé est ajouté à une liste selon un indexage en « column order » présenté dans le chapitre 2. La figure suivante montre un exemple de déroulement de notre algorithme dans le cas d'une

hauteur égale à 2 et une largeur égale à 3. Les points noirs représentent les variations du point à l'origine.

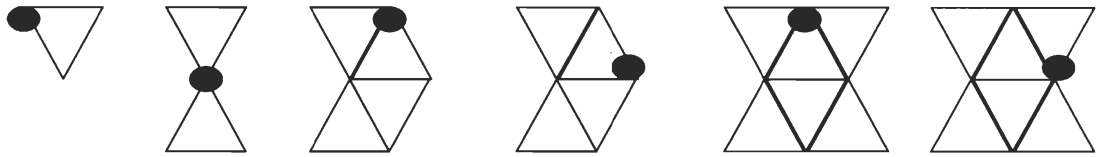


Figure 4.13 Déroulement de l'algorithme dans le cas de l= 2 et h = 2

4.3.2.2 Partition d'un triangle

Une fois que nous avons créé nos triangles de plus haut niveau, il ne reste plus qu'à les partitionner en d'autres triangles de tailles inférieures. Notre approche considère chaque triangle de plus haut niveau comme un sous-espace élémentaire et elle le décompose par la suite. L'ensemble de ces triangles décomposés forme notre espace total. Toujours dans l'optique d'une approche « top-down », notre algorithme de partition est un algorithme récursif qui vise à diviser les cellules. Il est formé de deux parties :

La première est la fonction qui fait appel à la fonction récursive. Cette première fonction, qui est formée par une boucle Pour, qui parcourt la liste des triangles, obtient un triangle et le passe comme paramètre à la fonction récursive. Elle admet comme paramètre un nombre représentant le facteur de résolution, ou encore le nombre de niveaux par triangle. L'algorithme est le suivant :

```
PartitionTriangle( Entier resolution )
Début
        Liste temp
        Pour  i = 0; i < Taille ListTriangle,  i++

            Triangle T = List ( i )
            Temp = vide ;
            PartTriangle(T, resolution, 1, temp);
            Ajouter la liste temp au Triangle T

        FinPour
Fin
```

Figure 4.14 Algorithme de partitionnement de l'espace

Il est à remarquer que rien ne nous empêche d'exécuter l'algorithme de partition d'un triangle après l'avoir créé lors de la partition de l'espace. Toutefois, ceci est peu performent à cause de la récursivité dans une boucle imbriquée. C'est pour cette raison que nous avons préféré l'exécuter à la suite de la division de notre espace. La deuxième fonction est la fonction récursive proprement dite. Elle admet comme paramètres le triangle T à partitionner, le facteur de résolution et une liste vide pour stocker les nouveaux triangles créés et les attribuer au grand triangle initial. (Voir figure 4.16).

Cet algorithme fonctionne comme suit : premièrement, il vérifie le cas trivial à chaque fois : le nombre de divisions par triangle doit être inférieur ou égal au facteur de résolution. Si cette condition est vérifiée, il prend les coordonnées du triangle en considération et il les divise par deux pour obtenir des triangles plus petits. Ensuite, il affecte les points fraîchement calculés aux anciens points du triangle précédant et l'appel récursif continue de s'exécuter jusqu'à ce que le cas trivial soit satisfait. Notons que la division se fait récursivement pour les 4 triangles enfant à chaque passe et non pas pour un seul. Aussi, à la première passe, on découpe le grand triangle initial, ensuite les 4 triangles résultant de cette passe sont à leur tour divisés en 4 autres sous-triangles plus petits. On continue ainsi jusqu'à ce que la condition d'arrêt soit vérifiée. Notre algorithme ne tient pas compte de l'orientation du triangle. Qu'il soit orienté vers le haut ou vers le bas, il va s'exécuter de la même façon. Ceci est un autre facteur de généricité en plus de pouvoir configurer le partitionnement selon le nombre de niveaux et la taille. L'exécution de notre algorithme pour un triangle orienté vers le haut et un facteur de résolution égal à 2 est montrée par la figure suivante.

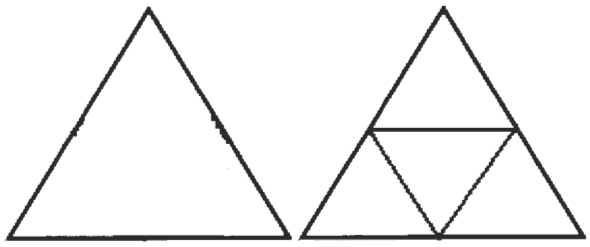


Figure 4.15 Résultat de l'algorithme de partition pour un facteur de résolution égal à 2

```
PartTriangle(Triangle T, resolution, nbfois, List )
  Début
        Si ( nbfois <= res)
           // Calcul des nouveaux points
            Point  G = (0, 0) , Point  D = (0, 0) ,Point Ha = (0, 0)
            Tab[] = Trier les points du triangle T
            xG  = ( Tab[ 0 ].X ) + ( Tab[ 1 ].X) )/2 ,  yG=  (Tab [ 0 ].Y) + (Tab[ 1 ].Y) / 2
            xD =  (Tab [ 2 ].X) + ( Tab[1 ].X )/ 2 , yD =  (Tab [ 2 ] .Y) + (Tab[ 1 ].Y) / 2
            axH = (Tab [ 0 ].X) + (Tab[ 2 ].X)/ 2 ,ayH= (Tab  [ 0 ].Y) + (Tab[ 2 ].Y) / 2

           // Calcul des points du triangle du milieu
           G = (xG,yG) ,  D = ( xD,yD ) ,  Ha =(xHa,yHa)
            TabM  [ ]  =  G, D, Ha
            Triangle TM = tabM
            TabM = Trier les points du triangle TM
            Ajouter à List le triangle  TM  et le dessiner

           // Calcul des points du  triangle de Gauche
            G = Tab[ 0 ] , D = TabM [ 0 ], Ha = TabM [ 2 ]
            tabG = G, D, Ha
            Triangle TG = tabG
            Ajouter à List le triangle  TG  et le dessiner

           // Calcul des points du   triangle de Droite
            G = TM[ 1 ] , D = T[ 1 ] , Ha = TabM[ 2 ]
            TabD [ ] = G, D, Ha
             Triangle TD = TabD
            Ajouter à List le triangle  TD  et le dessiner

           // Calcul des points du   triangle en Haut
            G = TabM 0
            D = TabM 1
            Ha = T 2
            [] tabH = G, D, Ha
            Triangle TH = tabH
           Ajouter à List le triangle  TH et le dessiner
            nbfois = nbfois + 1
            PartTriangle(TM, res,nbfois, List)          Appel récursif
            PartTriangle(TG, res, nbfois, List)         Appel récursif
            PartTriangle(TD, res, nbfois, List)         Appel récursif
            PartTriangle(TH, res, nbfois, List)         Appel récursif


        FinSi
Fin
```

Figure 4.16 Algorithme récursif pour partitionner un triangle en plusieurs sous-triangles

Notons qu'une liste des sous-triangles est mise à jour à chaque passe de l'algorithme. À la fin, le triangle de plus haut niveau est associé à une liste contenant l'ensemble de ses enfants. Ceci nous permet un accès direct en temps constant à la liste, à l'aide d'un mécanisme d'indexage que nous présenterons ultérieurement.

Après avoir partitionné notre espace, puis divisé les triangles en sous-triangles et ajuster les listes d'indexation, il ne reste plus qu'à former des cellules hexagonales hiérarchiques. Ceci est présenté dans la section suivante.

4.3.3 Grilles Hexagonales

Nous présentons ici les algorithmes qui ont servi à créer des grilles hexagonales.

4.3.3.1 Algorithmes de construction d'une grille hexagonale

Nous présentons notre algorithme *BuildHexag* qui permet de construire une cellule hexagonale à partir de six triangles équilatéraux.

```
BuildHexag ( Liste list )
Début
        idxS = 3
        idxSB = 0;
        i = 0 j = 0; cpt = 0;
        x = taille de la liste
        nextCol = 0;
        start = idxS;
        itr = 1;
        Tantque ( cpt <  largeur )
            Tantque ( i < hauteur )
                    courant = Créer Le demi hexagone d'en haut
                    idxSB = GetPosNextt(idxS, courant);
                       Si (i == 0)
                            nextCol = idxSB + 3                  // la prochaine colonne
                              courant = Créer Le demi hexagone d'en haut
                            idxSB = Trouver le prochain point du triangle en bas
                            idxS = idxSB;                 // Commencer au point le plus bas
                            H1 =  Créer un hexagone à partir des deux moitiés calculées
                            Ajouter H2 à list
                            i++; itr++;
                        FinSi
                  FinTantque
                   idxS = nextCol;                                // traiter la prochaine colonne
                    idxSB = 0;
                Tantque (j < hauteur)
                          ArrayList lis = new ArrayList();
                           t = Créer Le demi hexagone d'en haut
                           idxSB = Trouver le prochain point du triangle en bas
                           t = Créer le demi hexagone d'en bas
                           idxSB = Trouver le prochain point du triangle en bas
                          Si  ( idxSB != 0  )
                              idxS = idxSB;
                           FinSi
                          H2 =  Créer un hexagone à partir des deux moitiés calculées
                          Ajouter H2 à liste
                           j++; itr++
                      FinTantque
                        i = 0; j = 0;
                       start = start + 6;
                        idxS = start;
                       cpt = cpt + 1;
            FinTantque
Fin
```

Figure 4.17 Algorithme de création de grilles hexagonale

Cet algorithme admet comme paramètre d'entrée une liste. Cette dernière va contenir les enfants relatifs à chaque triangle de plus haut niveau. Ainsi, nous allons avoir une structure linéaire contenant les objets parent, suivis de leurs enfants. Pour construire un hexagone à partir des triangles, nous savons qu'un hexagone est formé de deux parties. Une partie supérieure contenant 3 triangles équilatéraux, et une partie inférieure qui lui est symétrique par rapport à l'axe des X. Ainsi, il suffit juste de rassembler la partie supérieure et inférieure dans une seule entité. Pour ce faire, on définit deux variables qui représentent les index des cellules triangulaires appartenant à chaque partie, c'est-à-dire la partie d'en haut et celle d'en bas. Ce sont ces variables qui changeront lors du déroulement de l'algorithme pour caractériser une cellule donnée à un instant donné. On définit la variable (*nextcolumn*) qui représentera la prochaine colonne à traiter. De plus, bien que les triangles soient partitionnés selon un « Column ordering », les sous-triangles sont désordonnés à cause de la récursivité. Il faut donc les ordonner selon une courbe de remplissage de l'espace. Nous avons choisi le « Row-order ». Il a fallu donc les organiser selon ce mode d'indexage. Ceci a été fait au niveau du prétraitement. Pour revenir à l'explication de l'algorithme, l'idée générale est la suivante :

Tant que les limites de la grille ne sont pas atteintes, il faut fixer le premier triangle d'en haut T, créer le demi-hexagone d'en haut, trouver le prochain triangle d'en bas qui est symétrique à T et créer le prochain demi-hexagone d'en bas. Ensuite, il faut former une seule cellule hexagonale à partir des deux moitiés fraîchement créées et l'ajouter à la liste globale. Finalement, incrémenter les compteurs et surtout la variable *nextColumn* pour pointer sur la colonne suivante. Tout ce processus se fait à l'intérieur de boucles imbriquées. Il est à noter que nous n'avons pas établit une équation mathématique pour incrémenter les index de manière formelle, afin que les hexagones ne se chevauchent pas. Nous avons utilisé une approche heuristique pour trouver les bons chiffres. Il est aussi important de remarquer que lors du premier test de cet algorithme les résultats étaient médiocres. En effet, il était peu rapide et avait une complexité cubique. En fait, le handicap réside dans la recherche du prochain index du prochain triangle de la moitié d'hexagone d'en haut ou d'en bas. Certes, au départ pour trouver le triangle qu'on cherchait, on faisait un parcours linéaire de la liste entière des sous-triangles. En connaissant le père et ses fils, on ne fait une recherche que dans la liste des fils, qui sont de l'ordre de quelques centaines

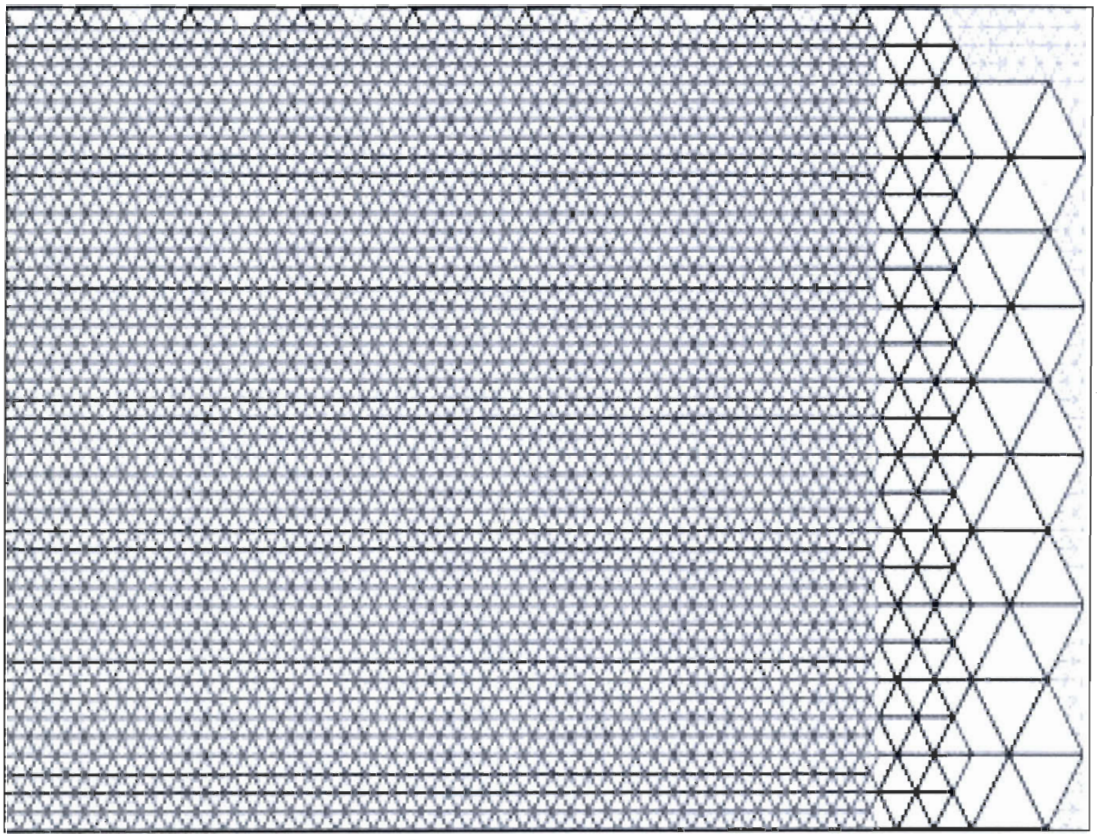
au pire des cas. Ceci est également peu performant car en moyenne, on sera obligé de parcourir la liste des triangles de plus haut niveau. Idéalement, il est convenable d'avoir un temps constant par le biais de l'index, autrement dit, à partir de l'index de la cellule courante, trouver l'index de la prochaine cellule. Pour ce faire, nous avons utilisé un algorithme des k-neihgbors (Samet,1990). L'idée est simple: lors d'une passe donnée, au lieu de visiter une cellule donnée, il faut visiter les k voisins de cette même cellule. Dans notre cas, connaissant l'index de départ, il suffit juste d'extraire les k cellules à visiter de la liste globale. Comme nous l'avons indiqué plus haut, nous commençons toujours par le triangle en haut et nous cherchons ensuite le triangle qui lui est symétrique et se trouvant en bas. De plus, à cause d'un partitionnement en « row-order », il est facile de déterminer l'intervalle de l'index recherché. Celui-ci est égal à l'index de la cellule courante additionné à la largeur de la grille. Donc, pour ne pas tomber dans un tâtonnement à cause des irrégularités des grilles sur les bords, nous avons retenu l'intervalle maximum contenant les k cellules voisines, et nous avons choisi de le parcourir avec un pire cas qui ne surgit que sur les limites de la grille. Toutefois, si on avait choisi un « column-order » comme courbe de remplissage de l'espace, lors de l'organisation des sous-triangles, nous aurions eu le même problème, mais transposé en fonction de la hauteur de la grille.

Pour tester l'efficacité de notre algorithme avant et après l'introduction de cette approche, nous avons fait un petit test de performance. Nous avons ajouté un compteur à la fin de chaque instruction dans le code afin de déterminer le nombre total d'instructions effectué. Nous sommes passé de 40000 vers 1000 instructions pour une hauteur égale à 250 et une largeur égale à 50. Pour conclure avec une approche « essai-erreur », nous avons pu trouver les bonnes constantes à ajouter aux incréments afin d'empêcher que les grilles ne se chevauchent. Nous avons organisé les sous-triangles selon un « row ordering » pour avoir un bon temps d'accès lors de la formation des grilles, en dépit du temps consacré à ce prétraitement. Finalement, nous nous sommes inspiré de l'algorithme des k-neibghors (Samet,1990) pour accélérer notre algorithme.

4.3.3.2 Algorithmes de construction des grilles hexagonales hiérarchiques

Il suffit d'appliquer l'algorithme précédant à chaque niveau de la hiérarchie et maintenir les grilles hexagonales fraîchement créées dans les bases de données. De plus, une fois que la construction de ces grilles hexagonales est terminée et enregistrée, leur affichage se fait

en un temps constant. En fait, la hiérarchie réside dans le fait que le facteur de résolution est divisé par 4 à chaque niveau. Ainsi, les dimensions de la grille restent les mêmes, mais le nombre de cellules enfants diminue. Plus on hiérarchise, plus ce nombre diminue. Les grilles sont enregistrées dans des tables différentes avec un index interne pour référencer les cellules de plus haut niveau entre elles. Ceci sera abordé dans la section 4.3.3.3. Par ailleurs, la hiérarchie réside dans le fait de créer des cellules hexagonales à partir des triangles équilatéraux, qui eux-mêmes peuvent composer une ou plusieurs cellules hexagonales. Contrairement à la cellule de la structure de GBT (Gibson, 1982) qui se décompose en un nombre limité de sous-cellules hexagonaux de tailles inférieurs. C'est là où réside toute l'innovation de notre approche. La figure 4.18 montre une portion de notre grille.



4.18 Une vue des grilles hiérarchiques de 3 niveaux

4.3.3.3 Index de la structure

Afin d'établir un lien entre les différentes cellules de la structure, nous avons besoin d'un mécanisme d'adressage et d'un index. Ceci permettra de retrouver une cellule donnée, qu'elle soit triangulaire ou bien hexagonale, peu importe le niveau de résolution. Les tableaux 4.4 e 4.5 montrent un exemple de notre index.

IdHex	IdTriangle	IdxTriangle
210,3	3	10002
210,3	824	34444
210,3	150	45678
210,3	4	73567
210,3	825	56777
210,3	5	64784

Tableau 4-4 Index externe

IdxTriangle	Points
10002	(12,345),(45,987),(12,345)
34444	(12345,55),(45987,56),(12345,78)
45678	(12,35),(45,97),(125,99)
73567	(745,777),(987,66),(145,10)
56777	(125,56),(487,455),(12,50)
64784	(745,789),(987,22),(145,10)

Tableau 4-5 Index interne

Nous avons deux index : l'un est appelé index externe car il référence une cellule hexagonale d'une grille donnée à un niveau donné. Le deuxième est un index interne car il enregistre les coordonnées des points d'un triangle donné. Du fait que ces coordonnées existent déjà au niveau d'une liste, à la suite du partitionnement de l'espace (voir section 4.2.2), il n'est pas nécessaire de dupliquer l'information. C'est pour cette raison que nous n'avons pas un seul index contenant les identifiants des cellules et les coordonnées des points qui les forment. Au niveau de l'index externe, le champ *IdxTriangle* référence les coordonnées des triangles appartenant à un hexagone. Quant au champ *IdTriangle*, il indique les index des six triangles qui constituent un hexagone dans une grille donnée à un niveau donné. Finalement, le champ *IdHex*, est formé par un couple de nombres. Le

premier chiffre référence l'index de la cellule hexagonale en question. Le deuxième le niveau de résolution auquel elle appartient. C'est avec ce dernier que nous maintenons les liens hiérarchiques entres les différentes grilles. Ceci est à l'image de l'index de Samet (Samet,1990) qui sauvegarde dans sa structure un couple de chiffres binaires. Le premier représente le « path array » et le deuxième le niveau de profondeur dans l'arbre de hiérarchie, et par le fait même le niveau de résolution hiérarchique. Au sujet du champ *IdxTriangle*, il renvoie aux triangles issus d'une subdivision de l'espace. En fait, il établit un lien entre un hexagone et les points qui le forment. En somme, c'est avec tous ces index que nous référençons spatialement chaque cellule triangulaire ou hexagonale de notre grille. Remarquons que c'est une structure qui ressemble aux tables des bases de données. Ceci facilite le lien avec les autres données géographiques exportées dans notre système, par le biais des champs cités plus haut qui serviront de clés étrangères dans les autres tables. Ceci sera détaillé quand nous présenterons l'architecture de notre système dans un chapitre 6.

4.4 Conclusion

Dans cette section, nous avons présenté notre approche et notre solution finale afin de résoudre une grande partie de notre problématique. Nous avons présenté également, nos premiers essais qui n'ont pas été concluants. Toutefois, c'est à la lumière de ces essais, que nous avons pu arriver à notre solution. Cette dernière est générique puisqu'elle permet de construire la grille selon différents paramètres tels que les dimensions, le facteur de résolution et la taille d'une cellule. De plus, elle permet de maintenir une relation hiérarchique par l'intermédiaire d'un mécanisme d'indexage. Dans les chapitres suivants nous allons appliquer ces grilles dans un contexte d'analyse spatiale et de géostatistique afin de tirer profit des avantages que nous venons de citer.

Chapitre 5 : Analyse spatiale et préparation du jeu de données

5.1 Introduction générale

Dans cette introduction nous présentons le contexte général dans lequel s'inscrit notre projet et l'analyse que nous avons élaborée. Nous présentons les données utilisées, les objectifs de notre analyse et les hypothèses que nous avons émises.

5.1.1 Contexte général

C'est dans le cadre du projet *MUSCAMAGS*[3] *« Multi-scale multi-agent geo-imulation »*, que cette analyse a été effectuée. Ce dernier est un projet de recherche qui vise à fournir aux décideurs des outils logiciels de géo-simulation afin de supporter leur processus de prise de décision, et ce dans plusieurs domaines tels que le domaine militaire, médical, industriel, ou de transport. Justement, c'est dans ce dernier domaine que s'inscrit cette étude. En effet, bien que les prédictions en 2001 aient annoncé une circulation plus fluide au cours des années à venir et une baisse possible du nombre de voitures sur les routes de la ville de Québec, des observations réelles de ces routes ont montré le contraire. Ces observations se sont confirmées au cours du temps, c'est-à-dire entre 2001 et 2006, sans connaître les raisons exactes qui provoquent ce phénomène géographique. C'est exactement dans ce contexte que se place l'analyse spatiale illustrée dans ce chapitre. Pour ce faire, il nous faut des données qui reflètent de manière représentative les acteurs de ce phénomène : en particulier, la population de la ville de Québec et des informations sur le réseau routier de la ville elle-même. Ainsi, nous utilisons deux types de données : des données thématiques ou statistiques (données issues de la simulation des déplacements) et des données spatiales et cartographiques (grilles et réseaux routiers de la ville de Québec). Les sections suivantes décrivent ces données.

[3] http://www2.ift.ulaval.ca/muscamags/

5.1.2 Données de l'analyse

Dans cette section, nous présentons succinctement la nature des données utilisées pour élaborer cette analyse ainsi que la démarche utilisée.

5.1.2.1 Données thématiques

Les données thématiques ne sont autres qu'une population synthétique qui est créée dans le cadre d'un projet de doctorat (Chaker, 2009). Dans ce projet on simule les déplacements des personnes de manière individuelle ou collective à l'échelle de la ville de Québec. Afin de simuler cette dynamique urbaine, il est nécessaire d'estimer la composition des ménages en individus, leurs caractéristiques et leur répartition dans l'espace. C'est à l'aide de l'enquête OD de 2001 que les caractéristiques de chaque ménage de la ville de Québec, ont pu être déterminées. Les données de la population synthétique sont enregistrées dans une base de données contenant plusieurs tables, les plus importantes concernent les personnes et leurs occupations au sein d'un ménage donné, leur mode de déplacement et le type des ménages auquel elles appartiennent. .

5.1.2.2 Données spatiales

Les ménages de cette population sont localisés sur une grille hexagonale et une carte routière qui couvrent tout l'espace géographique de la ville de Québec. Par conséquent, pour chaque ménage, on peut déterminer la position géographique de son lieu de résidence et la voir sur une carte. Ensuite, pour les fins de notre projet, le même processus a été répété, mais plutôt que de localiser les ménages sur une grille hexagonale, ils ont été localisés sur notre grille triangulaire. Les sous-sections suivantes décrivent les grilles et les cartes que nous utiliserons. Notons que nous présenterons les fichiers cartographiques par la suite.

La grille hexagonale est un ensemble de cellules hexagonales en mode vectoriel qui sont développées par le *CRAD*[4] *« Centre de recherche en aménagement et en développement »*. Chaque cellule hexagonale de cette grille, a 250 mètres de diamètre. Chaque cellule couvre un espace géographique qui contient des ménages. Chaque cellule possède également un centroide auquel on associe des informations agrégées comme par exemple la moyenne de ménages par cellule.

[4] http://www.crad.ulaval.ca/

Par ailleurs, une nouvelle grille triangulaire à été créée par une étude de maîtrise (Drolet, 2009) pour adapter les données aux objectifs de notre projet, essentiellement d'avoir des cellules triangulaires pour couvrir l'espace, plutôt que des cellules hexagonales. Cette grille est identique à la grille hexagonale initiale à l'exception de la forme géométrique de ses cellules. La population synthétique a été référencée de nouveau sur cette nouvelle grille afin de s'adapter aux grilles que nous avons présentées dans le chapitre 4 et aux objectifs de ce mémoire.

5.1.3 Méthode d'analyse

Dans cette section, on présente les hypothèses posées et la démarche suivie pour tenter d'expliquer l'augmentation du trafic routier pour la ville de Québec, en utilisant notre approche.

Nous cherchons à localiser les zones géographiques potentiellement génératrices de véhicules qui peuvent encombrer le réseau routier. C'est-à-dire les régions qui engendrent un surplus de véhicules sur le réseau routier. Nous présentons nos hypothèses pour atteindre nos objectifs.

Dans un premier temps, on s'intéresse à la population synthétique initiale. Dans un 2^{ième} temps, nous allons faire vieillir cette population afin de déterminer son impact sur le réseau routier au cours du temps. Pour chaque étape, nous avons besoin d'indicateurs clés et d'hypothèses pour cerner le phénomène que nous étudions, c'est-à-dire l'augmentation du trafic routier. Tout d'abord, nous allons présenter la définition des indicateurs que nous avons considérés.

On appelle :

- *Conducteur potentiel* : Toute personne ayant l'âge de conduire (16 ans et plus) et qui ne possède pas encore de permis de conduire.
- *Revenu par ménage* : Le revenu annuel d'un ménage.
- *Nombre de déficit de voiture par ménage* : Les ménages qui ont besoin d'un ou de plusieurs voitures en tenant compte du nombre de conducteurs potentiels et du nombre du nombre de voitures effectivement possédés par les ménages.
- *Catégorie de déficit* : Le nombre de voitures manquant par ménage par rapport au nombre de conducteurs potentiels par ménage.

- *Zones génératrices de déficits* : Un espace géographique ou un ensemble de cellules contenant plusieurs ménages en déficit de véhicules et chaque ménage appartenant à une catégorie de déficit.

Nous supposons que :

- La sous-population d'individus susceptibles d'encombrer le réseau routier correspond particulièrement aux étudiants qui ont besoin d'un véhicule pour se rendre à leurs lieux d'études (Université, CÉGEP).
- Une famille avec un revenu élevé ou moyen pourrait acquérir une ou plusieurs voitures pour leurs enfants en âge de conduire.
- L'âge moyen pour commencer à aller à l'université est 20 ans.

Dans un deuxième temps, nous faisons évoluer la population initiale en faisant vieillir les jeunes sur six années, afin d'étudier l'apparition des nouveaux conducteurs potentiels. De plus, pour chaque cas d'étude nous allons prendre en considération le revenu par ménage. À cet effet, nous émettons les hypothèses simplificatrices suivantes :

- La taille de la population de Québec est stable. C'est-à-dire, on ne considère pas l'évolution démographique (naissances et décès).
- La migration d'un ménage vers un autre domicile, ne va pas influencer l'achat d'un véhicule.
- Le sexe d'une personne n'influence pas l'acquisition d'un véhicule.

À partir de toutes ces hypothèses nous allons tenter de localiser les zones génératrices de déficit de véhicules entre 2001 et 2006 et par le fait même préparer un jeu de données propice à notre prototype. En effet, les données descriptives serviront de batterie de test pour notre outil. Dans ce chapitre, nous présentons notre démarche d'analyse et les résultats obtenus.

5.2 Population Initiale

5.2.1 Extraction et nettoyage des données

Il est à remarquer que ces données sont enregistrées dans une base de données. Nous allons appliquer des requêtes SQL afin d'extraire les données qui nous intéressent. L'organisation de ces données est présentée par la figure D.1 de l'annexe D.

5.2.1.1 Données thématiques

Le but est de trouver le déficit en nombre de conducteurs par ménage par rapport au nombre de voitures disponibles. Pour ce faire, nous sommes partis de l'hypothèse suivante : Le nombre des conducteurs par ménage est le même que le nombre de permis de conduire par ménage. Après cela, nous avons extrait une table contenant l'âge, le sexe, une référence spatiale, l'identificateur du ménage pour les personnes possédant des permis de conduire et celles qui ne l'ont pas. Le résultat final est stocké dans la table *DonneesPermisFines* grâce à la requête SQL suivante :

```
SELECT Personnes01.UniqueMena, Personnes01.Sexe, Age, Personnes01.PermisCond FROM
Personnes01 WHERE Personnes01.PermisCond=1 Or Personnes01.PermisCond=2;
```

Ensuite. Nous avons calculé le nombre total des personnes qui ont un permis de conduire par ménage à partir de la table précédente. Ainsi, nous obtenons, le nombre total des conducteurs. Le résultat est enregistré dans la table *NombreConducteurMenage* par le biais de la requête suivante :

```
SELECT UniqueMena, Count (*) AS NBCondM FROM PermisMFines GROUP BY (UniqueMena);
```

UniqueMena	PermisCon	Sexe
41	1	1
41	1	2

UniqueMena	PermisCon
41	2

Tableau 5-1 Un exemple de données initiales et agrégées

La prochaine étape consiste à trouver les ménages ayant des conducteurs potentiels. Un conducteur potentiel est une personne ayant l'âge de conduire (16 ans et plus) et qui ne possède pas de permis de conduire. Ceci peut être reformulé comme suit : un conducteur potentiel est une personne dont la catégorie d'âge est égale à 4 et qui ne possède pas de permis de conduire. Effectivement, cette catégorie renvoie à la tranche d'âge de 15 à 19 ans inclusivement et la valeur du champ qui enregistre l'attribut *Permis de conduire* est égale à 2. Le résultat est sauvegardé dans la table *RangeUniqueMenage* à l'aide de cette requête :

```
SELECT DISTINCT UniqueMena, sexe, PermisCond, Age, LPasserBus, Occupation
FROM Personnes01 WHERE age=4 And (PermisCond=2);
```

Une fois que nous avons identifié les ménages qui ont des conducteurs potentiels, nous pourrons calculer le nombre d'étudiants par ménage. En effet, dans un ménage donné, il peut y avoir plusieurs étudiants qui sont en âge de conduire et qui ne possèdent pas un permis de conduire. Ainsi à partir de la table précédente, nous avons calculé le nombre total des conducteurs par ménage, que l'on sauvegarde dans la table *NBetudiantMenage*. Ceci se fait avec cette requête de regroupement suivante :

```
SELECT UniqueMena, Count(*) AS NbEtudMenage FROM RangeUniqueMenage
GROUP BY (UniqueMena) HAVING Count(*)>1 Or Count(*)>2 Or Count(*)=1;
```

Finalement on fait une jointure entre les tables *Menages01*, *personnes01*, *ConducteurPotentiel* et *NBEtudiantMenage*. En effet, cette jointure permet d'extraire tous les champs d'intérêts et de trouver le nombre total de conducteurs par ménage. Ce dernier n'est autre que la somme du nombre des conducteurs qui possèdent déjà un permis de conduire par ménage, et le nombre de conducteurs potentiels par ménage qui n'en possèdent pas. Notons, que pour les étudiants qui possèdent un permis de conduire, ils font déjà partie de la catégorie des conducteurs en possession d'un permis de conduire selon les données de la simulation. Ensuite, ces ménages sont enregistrés dans la table *NCTotal.* Jusqu'à ce stade, d'autres données ont été extraites et sauvegardées dans chaque table intermédiaire tels que l'âge, le sexe, l'occupation, l'index de la cellule triangulaire. La requête qui nous permet de faire cela est :

```
SELECT NBMConducteur.UniqueMena, ResidGrHex, RangeUniqueMenage.Age,
RangeUniqueMenage.sexe,RangeUniqueMenage.PermisCond,RangeUniqueMenage.LPasserBus,
RangeUniqueMenage.Occupation, NBMConducteur.NBCondM, NbAuto, NbEtudMenage,
(NBCondM+NbEtudMenage) AS TotalCondMenage FROM NBMConducteur, RangeUniqueMenage,
NBEtudMenage, Menages01 WHERE
NBMConducteur.UniqueMena=RangeUniqueMenage.UniqueMena, and
RangeUniqueMenage.UniqueMena=Menages01.UniqueMena And
RangeUniqueMenage.UniqueMena=NBEtudMenage.UniqueMena;
```

Enfin, on crée la table *ResultatDeficit* qui enregistre le déficit du nombre des conducteurs par ménage par rapport au nombre d'autos par ménage en effectuant la différence des deux champs qui les représentant respectivement.

```
SELECT UniqueMena, ResidGrHex, Age, Sexe, LPasserBus, Occupation, NbAuto,
NBCondM, TotalCondMenage, (TotalCondMenage-NbAuto) AS deficit FROM NCTotal;
```

5.2.1.2 Données Spatiales

Premièrement, nous avons préparé et intégré les données cartographiques et spatiales dans une seule base de données géographiques. Ces cartes sont : la carte du réseau routier, la grille triangulaire et la carte du réseau de transport publique par autobus de la ville de Québec. Chacune d'entre elle correspond à un « layer » dans le SIG *Mapinfo*. Ainsi, on peut afficher chaque carte toute seule ou les trois ensembles. Également, nous avons déterminé les couloirs d'autobus, c'est-à-dire les cellules de la grille triangulaire qui ont une intersection avec les lignes du réseau d'autobus de la ville de Québec. Ceci s'est effectué à l'aide du logiciel MapInfo et avec la requête spatiale suivante :

```
SELECT * FROM Grille_Triangles_Centres_Basse, lignesdebus
WHERE Grille_Triangles_Centres_Basse.obj Intersects lignesdebus.obj
```

Cette requête utilise l'operateur spatial « *Intersects* » propre au SIG qui opère sur les polygones de chaque table. La carte résultante sera utilisée par tout le processus d'analyse. La couleur en vert montre les couloirs d'autobus, celle en noir le réseau routier. La couleur bleu désigne les lignes du réseau de transport par bus. Les cellules de la grille sont en rose. La carte est montrée dans les figures 5.1a

Figure 5.1 Couloirs d'autobus de la ville de Québec

5.3 Analyse des données

La section précédente a montré comment nous avons préparé les différentes données descriptives propices à notre analyse. Dans cette section, nous présentons notre démarche d'analyse spatiale de ces données.

5.3.1 Catégories de déficit

Pour trouver les catégories de déficits de voitures par ménage, nous avons agrégé les données de la table *ResultatDeficit*. En effet, cette table enregistre plusieurs ménages par cellule. Par conséquent, un regroupement est nécessaire afin d'assigner une seule valeur pour tous les ménages qui appartiennent à une seule cellule. Pour trouver cette valeur, nous avons exploré deux approches. La première est basée sur un calcul de la moyenne de déficits des ménages par cellule. La deuxième est basée sur le calcul de la somme totale de déficits des ménages par cellule. Dans ce qui suit, nous présentons ces approches.

5.3.1.1 Calcul du déficit avec la moyenne

Lors d'un premier essai, nous avons calculé la moyenne de déficits de chaque ménage par cellule. Comme le montrent les deux tableaux suivants:

Données Déficit									
UniqueMena	ResidGrHex	Age	Sexe	LPasserBus	Occupation	NbAuto	NbCondM	TotalCondMenage	Deficit
44	21666	4	1	0	3	2	2	3	1
48	21666	4	2	0	3	1	2	3	2
53	21666	4	2	0	3	1	2	3	2
57	21666	4	1	0	3	1	2	3	2
62	21666	4	1	0	3	2	3	4	2
65	21666	4	2	0	3	2	3	4	2

Tableau 5-2 Déficit par ménage par cellule

Données Agrégées par moyenne						
ResidGrHex	NbMCel	NBECel	NbCondMCel	NbAutoCel	TotalCondMCel	deficitCel
21666	6	1	1,5	1,33333333333333	2,5	1,16666666666667

Tableau 5-3 Moyenne de déficit par ménage par cellule

Le tableau 5.2 contient les identificateurs des ménages et des cellules hexagonales. Il contient l'age et le sexe des individu des ménages, leurs occupations, le nombre de voiture (NbAuto) disponible par ménage , le nombre de conducteur par ménage, le nombre total des conducteurs par ménage et finalement le déficit en véhicules (Deficit).

Quant au tableau 5.3, il présente la moyenne de déficit par ménage par cellule. Il contient l'identificateur d'une cellule hexagonale. Le nombre de cellules, le nombre de conducteur par cellule et la moyenne de déficit par cellule.

Cependant, cette démarche n'est pas représentative, car le calcul de la moyenne a un effet de lissage sur les données et par conséquent, au niveau de la carte, le nombre des zones est faible. Pour créer la carte de densité (voir figure 5.2a et 5.2b), nous avons déterminé l'intervalle contenant la valeur maximale et la valeur minimale de déficits de voitures par ménage par cellule. Elles sont respectivement égales à 5 et 0. Par conséquent, chaque cellule possède 0, 1, 2, 3, 4 ou 5 déficits de voitures. Pour les représenter sur la carte, chaque catégorie est une table à part entière qui est importée dans *MapInfo*. Pour chaque table on affiche les cellules qui correspondent aux identifiants des références spatiales à l'aide de la requête spatiale suivante :

```
Select* from grille where Grille.CelID = CategorieDeficit.CelID
```

La figure 5.2a montre la carte de densité qui est le résultat de cette démarche.

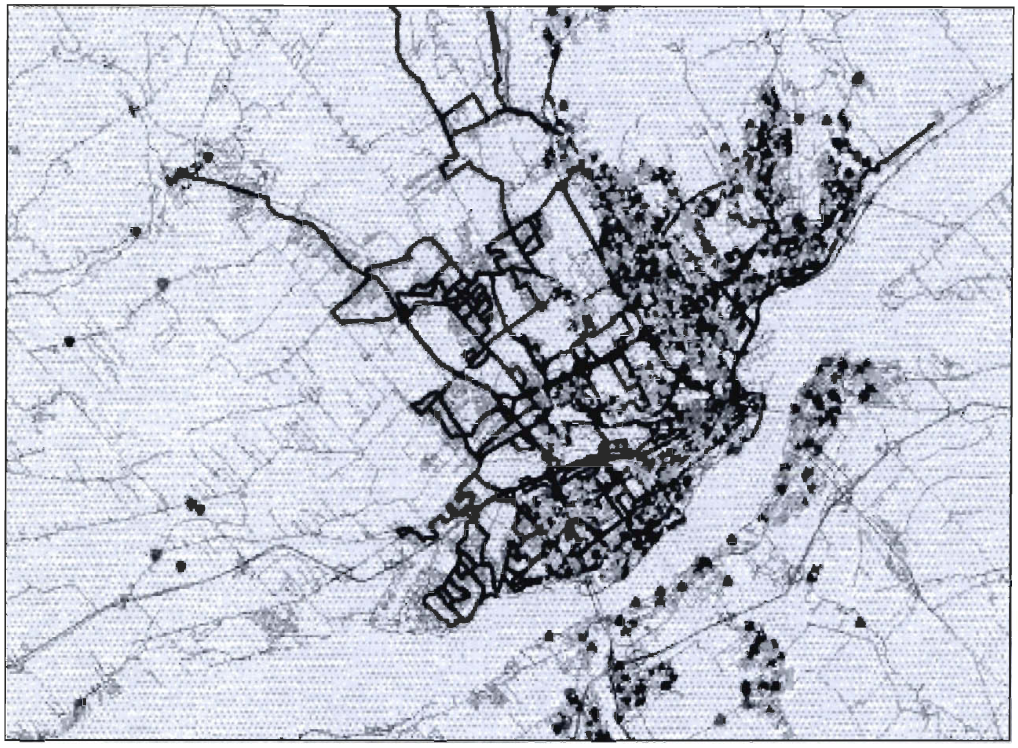


Figure 5.2 Vue élargie des cellules avec catégorie de déficit par moyenne

5.3.1.2 Calcul avec la somme

Lors d'un deuxième essai, nous avons calculé le nombre total de véhicules en déficit pour tous les ménages appartenant à la même cellule, dans le but de déterminer le nombre total de déficit de voiture par cellule. En effet, premièrement on additionne la somme du nombre de conducteurs potentiels par ménage par cellule avec la somme du nombre de conducteurs possédant déjà un permis de conduire par ménage par cellule. Nous obtenons ainsi le nombre total des conducteurs par cellule. Deuxièmement, on additionne le nombre de voitures par ménage par cellule. On obtient le nombre total de voitures par cellule. Finalement, la différence entre le nombre total de conducteurs par cellule et le nombre total de voitures par cellule donne le nombre total du déficit en voitures par cellule. Le tableau 5.4 montre un exemple de ces données qui sont agrégées par somme.

Données Agrégées par Somme						
ResidGrHex	NbMCel	NBECel	NbCondMCel	NbAutoCel	TotalCondMCel	deficitCel
21666	6	6	9	8	15	7

Tableau 5-4 Somme des déficits par cellule

La figure suivante montre la carte avec des catégories de déficit par la somme totale de déficits des ménages par cellule

Figure 5.3 Vue élargie de la carte de densité de ménages par cellule

La requête spatiale est la suivante :

```
Select * From GrilleTriangulaire where Grille.CelId = NombreMenaCel.CelId
```

La table *NombreMenaCel* est une table qui contient le nombre de ménages par cellule et qui est importée dans MapInfo. Elle compte le nombre des ménages localisés dans une même cellule à l'aide de l'opérateur *count* de SQL. Contrairement aux catégories de déficit par moyenne[5] où les valeurs varient de 0 à 5, les catégories de déficit par somme varient de 0 jusqu'à 51. Nous les avons classifiés par intervalles et avons choisi d'affecter 5 unités à chaque intervalle afin qu'on ait une échelle représentative. Notons que ces intervalles sont bornés à droite et ouverts à gauche. La table *catégorieDeficit* décrit ces catégories comme l'indique le tableau suivant :

[5] Du fait qu'on additionne les déficits de tous les ménages de la cellule, on obtient un nombre de cellules plus grand que dans le cas précédent où on affichait des catégories de déficit moyen par cellule.

Catégorie Déficit		
CategDid	Intervalles Catégories	Type Catégorie
0	0- 5	Catégorie de déficit entre 0 et 5 voitures
510	5-10	Catégorie de déficit 5 et 10 voitures
1015	10 - 15	Catégorie de déficit 10 et 15 voitures
1520	15 - 20	Catégorie de déficit 15 et 20 voitures
2025	20-25	Catégorie de déficit 20 et 25 voitures
2530	25-30	Catégorie de déficit 25 et 30 voitures
3035	30-35	Catégorie de déficit 30 et 35 voitures
3540	35-40	Catégorie de déficit 35 et 40 voitures
4045	40-45	Catégorie de déficit 40 et 45 voitures
4551	45- 51	Catégorie de déficit 45 et 50 voitures

Tableau 5-5 Les intervalles des catégories de déficit

Nous remarquons qu'avec un calcul par somme, nous obtenons des données plus fines. Nous avons donc écarté le calcul de déficit par moyenne des ménages par cellule et nous avons gardé cette solution. Dans ce qui suit, nous montrerons comment on peut localiser les ménages qui appartiennent à ces catégories de déficit de voitures.

Nous cherchons à trouver toutes les cellules ayant des ménages qui correspondent aux catégories précédentes (voir tableau 5.5). Pour ce faire, nous avons exécuté la même requête plusieurs fois. Pour chaque requête, le résultat est enregistré dans une table intermédiaire.

La routine SQL que nous avons utilisée, possède la même syntaxe, sauf que le paramètre *deficitCel* varie. En effet, on doit le majorer par la valeur minimale et maximale de chacun des intervalles des catégories de déficit précédentes. Par exemple, la requête suivante détermine les ménages qui appartiennent à l'intervalle [0,5].

```
SELECT ResidGrHex, CategDid, NBMCEL, NBECel, NBCondMCel, TotalCondMCel,
NbAutoCel, deficitCel FROM DonneesBAgregees, CategorieDeficit
WHERE deficitCel>= 0 And deficitCel <= 5 And CategorieDeficit.CategDId =
0;
```

Par la suite, nous avons fusionné les tables des cellules ayant des ménages pour chaque catégorie de déficit en une seule table, que nous avons appelée *CategorieDeficitM.* Cette dernière contient l'ensemble des cellules ayant des ménages en déficit de voitures.

```
Select * from CategorieDeficit where CouloirdAutobus.CelId =
CategorieDeficitM.ResidGrHex
```

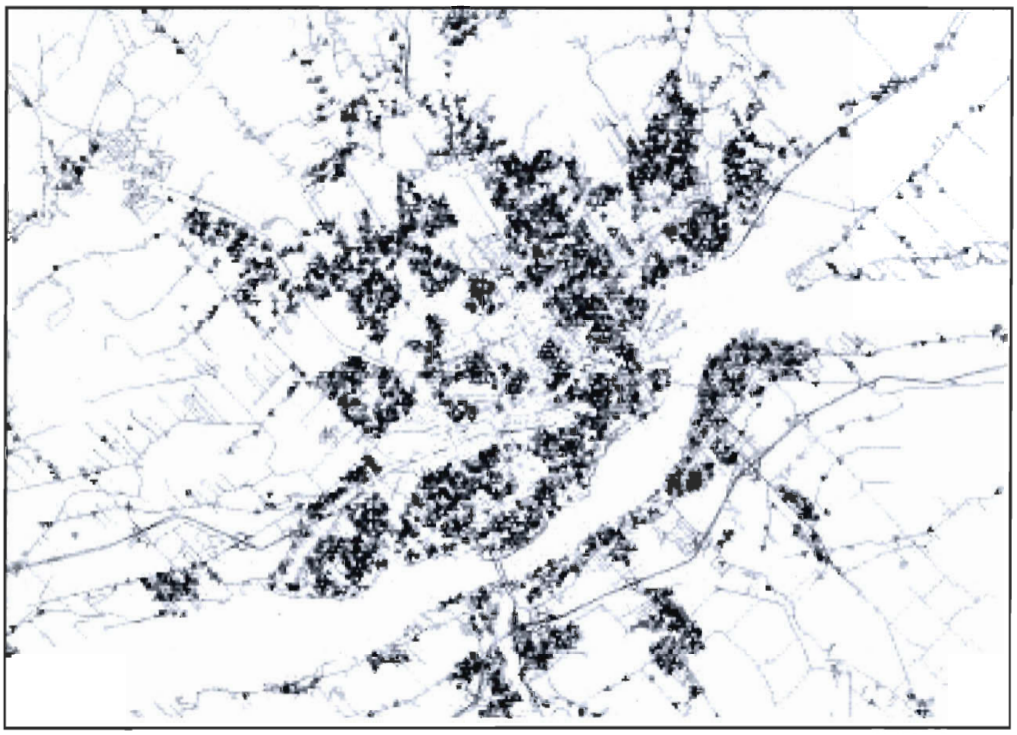


Figure 5.4 **élargie des cellules avec les catégories de déficit par somme**

Une carte thématique a été élaborée en proposant utilisant les codes sémiologiques suivants :

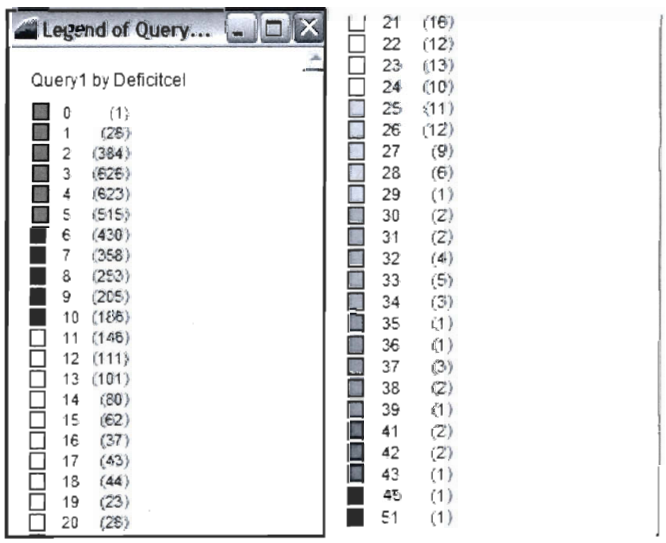


Figure 5.5 Légende des catégories de déficit

Remarquons, que dans la carte les couleurs dominantes sont le bleu pâle, le bleu foncé, le jaune et l'orange clair. C'est-à-dire, la majorité des cellules ont un déficit qui varie de 0 à 29 voitures.

5.3.1.3 Conclusion

Dans cette section nous avons considéré le déficit de voiture comme un premier critère d'analyse des données. Dans la section suivante, nous considérons un autre critère qui est le revenu par ménage afin de filtrer les données pour raffiner notre analyse.

5.3.2 Revenus par ménage

Comme nous l'avons mentionné précédemment, le revenu du ménage nous semble un facteur important pour déterminer le potentiel d'achat d'un véhicule pour un ménage appartenant à une catégorie donnée de déficit en véhicule. Toutefois, faute d'informations statistiques suffisantes (l'enquête OD ne compte pas l'information reliée aux revenus des personnes), nous avons opté pour une méthode qualitative. En effet, nous avons établi une liste de critères qui nous permettra d'évaluer si un ménage possède un revenu élevé, moyen ou faible. Ceci est possible à l'aide des tables *Occupation*, *Professions* et *TypeMenage* de l'enquête 01. La première décrit l'occupation de l'individu dans le ménage. Il peut être travailleur à temps complet, à temps partiel, aux études ou bien retraité. La deuxième, indique le type de profession exercée par les personnes qui travaillent. La troisième table montre le type de ménage, par exemple un ménage monoparental. Nous avons réalisé deux essais. Le premier est basé seulement sur les tables que nous venons de citer. Le deuxième se base sur une version plus récente de la table *Professions* (voir annexe A.1).

5.3.2.1 1^{ier} Essai

Pour construire notre liste, nous avons attribué des poids (*Fort, moyen, faible*) à chaque catégorie de professions. *Fort* pour les catégories 1,2 et 3, poids *Moyen* pour les catégories 4 et 6 et le poids *faible* pour les catégories 5 et 7. Ensuite, nous avons croisé les poids de chaque catégorie pour trouver l'ensemble des combinaisons possibles. Ainsi, on tient compte qu'il puisse y avoir deux personnes adultes qui travaillent dans un ménage ou une seule personne (ménage monoparental). La section 2.4 explique en détail cette démarche. En fait, nous avons 5 cas possible : (*Fort* x *Moyen*), (*Fort* x *Fort*), (*Fort* x *Faible*), (*Moyen* x *Moyen*) et (*Moyen* x *Faible*). Le résultat est 1: *Revenu fort* , 2: *Revenu moyen et* 3:

Revenu faible. Le tableau suivant montre le résultat de ces croisements et par le fait même, notre liste des critères.

Code Revenu 1	Code Revenu 2	Poids1	Poid2	Code RevenuTotal	Poid Revenu Total
1	5	Fort	Faible	1	Fort
2	5	Fort	Faible	1	Fort
3	5	Fort	Faible	1	Fort
1	7	Fort	Faible	1	Fort
2	7	Fort	Faible	1	Fort
3	7	Fort	Faible	1	Fort
4	5	Moyen	Faible	3	Faible
6	5	Moyen	Faible	3	Faible
4	7	Moyen	Faible	3	Faible
6	7	Moyen	Faible	3	Faible
4	6	Moyen	Moyen	2	Moyen
6	6	Moyen	Moyen	2	Moyen
4	4	Moyen	Moyen	2	Moyen
5	5	Faible	Faible	3	Faible
7	7	Faible	Faible	3	Faible
7	7	Faible	Faible	3	Faible
1	4	Fort	Moyen	1	Fort
2	4	Fort	Moyen	1	Fort
3	4	Fort	Moyen	1	Fort
1	6	Fort	Moyen	1	Fort
2	6	Fort	Moyen	1	Fort
3	6	Fort	Moyen	1	Fort

Tableau 5-6 Liste qualitative des revenus par ménage du 1ier essai

Cette liste est utilisée comme suit : pour un ménage donné de la table *personnes01*et pour chaque individu de ce ménage, on vérifie la combinaison des valeurs du champ *prof_type* de chaque individu et on cherche le poids du revenu qui correspond à cette combinaison dans notre liste. Notons qu'on ne peut pas raffiner plus, c'est-à-dire, on ne peut pas croiser des professions entre elles et aller à un niveau de détail plus fin, car pour un ménage donné nous avons le type de catégorie de profession, mais nous n'avons pas la profession en tant que telle des personnes; donc on ne peut pas retracer la profession exacte de chaque individu au sein d'un ménage.

5.3.2.2 2ième Essai

La classification précédente a été modifiée grâce à une table *Professions* plus récente que nous avons obtenue (voir annexe A.1), et dans laquelle les catégories présentées plus haut changent. En effet, nos catégories deviennent alors *fort* pour 1 et 3, *moyen* pour 2, 4,5,

6 et *faible* pour 7. Cette liste (voir tableau 5.8) est enregistrée dans la table *RevenuMenage.* Ceci est à cause des catégories d'emplois dans la table récentes. Ils catégorisent les professions de manière différente et plus récentes. En effet, dans la table référence des professions d'emplois de l'annexe A.2, nous remarquons que les professions qui peuvent avoir un salaire élevé comme directeur, ingénieur ou médecin sont dans les catégories 1 et 3. Ceux qui ont des salaires moyens (technicien, Secrétaire etc.) appartiennent aux catégories 4, 5 et 6. Finalement, les métiers à faible revenu sont dans la catégorie 7. C'est pour ces raisons que nous avons adopté cette nouvelle classification représentée par le tableau 5.7

Code Revenu 1	Code Revenu 2	Poids1	Poid2	Code RevenuTotal	Poid Revenu Total
1	5	Fort	Moyen	1	Fort
2	5	Moyen	Moyen	2	Moyen
3	5	Fort	Moyen	1	Fort
1	7	Fort	Faible	1	Fort
2	7	Moyen	Faible	2	Moyen
3	7	Fort	Faible	1	Fort
4	5	Moyen	Moyen	2	Moyen
6	5	Moyen	Moyen	2	Moyen
4	7	Moyen	Faible	2	Moyen
6	7	Moyen	Faible	2	Moyen
4	6	Moyen	Moyen	2	Moyen
6	6	Moyen	Moyen	2	Moyen
4	4	Moyen	Moyen	2	Moyen
5	5	Moyen	Moyen	2	Moyen
7	7	Faible	Faible	3	Faible
1	4	Fort	Moyen	1	Fort
2	4	Moyen	Moyen	2	Moyen
3	4	Fort	Moyen	1	Fort
1	6	Fort	Moyen	1	Fort
2	6	Moyen	Moyen	2	Moyen
3	6	Fort	Moyen	1	Fort
3	2	Fort	Moyen	1	Fort
1	1	Fort	Fort	1	Fort
3	3	Fort	Fort	1	Fort
1	3	Fort	Fort	1	Fort
2	2	Moyen	Moyen	2	Moyen
1	2	Fort	Fort	1	Fort

Tableau 5-7 Liste qualitative des revenus par ménage du 2ième essai

5.3.3 Travailleurs par ménage

Après avoir déterminé notre table de revenu, nous devons trouver les travailleurs par ménage dont le revenu appartient à la table des revenus et ce pour les différents types de ménages. En fait, on chercher à cerner le revenu des ménages qui sont en déficit de voiture afin de déterminer s'ils sont capables d'en acquérir un ou plusieurs véhicules. Cependant, il existe trois types de ménages :

- Un ménage avec deux travailleurs de sexe opposés.
- Un ménage avec un seul travailleur de sexe féminin ou masculin.
- Un ménage avec deux travailleurs de même sexe.

On va s'intéresser seulement au cas a et b, car les ménages de même sexe sont considérés en nombres négligeables.

5.3.3.1 Ménages avec deux travailleurs

Nous avons extrait les ménages ayant deux personnes qui travaillent de sexe masculin et de sexe féminin dans deux tables intermédiaires *ProfSexe1* et *ProfSexe2* avec les requêtes suivantes :

```
SELECT uniqueMena, uniquePers, prof_type, sexe, age, occupation
FROM personnes01 WHERE sexe=1 And occupation=1;

SELECT uniqueMena, uniquePers, prof_type, sexe, age, occupation
FROM personnes01 WHERE sexe=2 And occupation=1;
```

Ensuite, nous avons fait une jointure des deux tables précédentes sur le champ *UniqueMena* pour trouver les personnes qui travaillent, qui sont de sexes opposés et qui appartiennent au même ménage. Ainsi, on obtient les ménages ayant deux travailleurs de sexe masculin et de sexe féminin. La table résultante est *ProfFBiTravailleurs.* La requête est la suivante :

```
SELECT ProfSexe1.UniqueMena AS UM1, ProfSexe2.UniqueMena AS UM2, ProfSexe1.age,
ProfSexe1.sexe, ProfSexe2.sexe, ProfSexe1.prof_type AS prof_type1,
ProfSexe2.prof_type AS prof_type2 FROM ProfSexe1, ProfSexe2
WHERE ProfSexe1.UniqueMena=ProfSexe2.UniqueMena;
```

Également, nous avons cherché le revenu de ces ménages en nous basant sur notre liste qualitative de revenus (voir section 2.3). Nous avons ajouté les champs *RevenuTotal* et *poid*

revenuTotal. Le premier indique le revenu total par ménage (1,2,3) et le deuxième indique son poids (*fort*, *faible,moyen*). Le résultat est la table *RevenuTotalMenag.* Les requêtes responsables de cette opération sont :

```
SELECT [ UM1] , age, prof_type1, prof_type2, CodeRMenage, PoidRevenuTotal
FROM RevenuMenage, ProfFBiTravailleurs WHERE CodeRConjoint1=prof_type1
And CodeRConjoint2=prof_type2;
```

```
SELECT ResidGrHex, RTotalMenage.UM1, age, prof_type1, prof_type2,CodeRMenage,
PoidRevenuTotal FROM Menages01, RTotalMenage
WHERE Menages01.UniqueMena = RTotalMenage.UM1;
```

Un exemple de la table résultat *RevenuTotalMenage* est le suivant :

RTotalMenage

UM1	age	prof_type1	prof_type2	CodeRMenage	PoidRevenuTotal
64	9	2	4	2	M
70	7	3	5	1	Fort

Tableau 5-8 Revenu d'un ménage

Pour le ménage 64, il y a deux travailleurs dont les types des professions sont 2 et 4. Le revenu total du ménage est 2, c'est-à-dire *moyen*. De plus, il faut faire une autre jointure avec la table *ResultatDeficit*, puisqu'on s'intéresse au revenu des ménages ayant un déficit de véhicules par l'entremise de la requête suivante.

```
SELECT ResidGrHex, ResultatDeficit.UniqueMena, ResultatDeficit.age, Sexe,
LPasserBus, Occupation, NbEtudMenage, prof_type1, prof_type2, CodeRMenage,
PoidRevenuTotal, NbAuto, NBCondM, TotalCondMenage, deficit
FROM ResultatDeficit, RTotalMenage WHERE
ResultatDeficit.UniqueMena=RTotalMenage.UM1;
```

Finalement, nous avons agrégé les données et nous les avons enregistrées dans la table *DonnéesRAgg.* Par conséquent, nous avons obtenu une seule valeur représentative par cellule. Cette dernière représente le potentiel d'achat de véhicule par cellule et elle est obtenue en additionnant les différents codes du revenu par ménage.

```
SELECT ResidGrHex, count (UniqueMena) AS NbMCel, sum (NbEtudMenage) AS NBECel,
sum(NbCondM) AS NbCondMCel, sum(NbAuto) AS NbAutoCel, sum(TotalCondMenage) AS
TotalCondMCel, sum(Deficit) AS deficitCel, sum(CodeRMenage) AS PotentielAchat
FROM ResultatDeficitRevenu GROUP BY (ResidGrHex) HAVING count (*)>1;
```

On obtient des valeurs qui varient de 2 à 28. On utilise la même technique que celle dans la section 2.2 pour majorer cet intervalle. Ainsi, on divise ce dernier sur 3 intervalles de neuf unités chacun. À chaque sous-intervalle, on associe un poids de revenu.

- De 2 à 10 : revenu faible.
- De 11 à 19 : revenu moyen
- De 20 à 28 : revenu fort.

Pour finir, il ne reste plus qu'à intégrer cette table dans *MapInfo* et exécuter la requête spatiale suivante :

```
Select * from Grille_Triangles_Centres_Basse, DONNESBR
Where Grille_Triangles_Centres_Basse.ID = DONNESBR.Residgrhex
```

On obtient alors la carte de la figure 5.6.

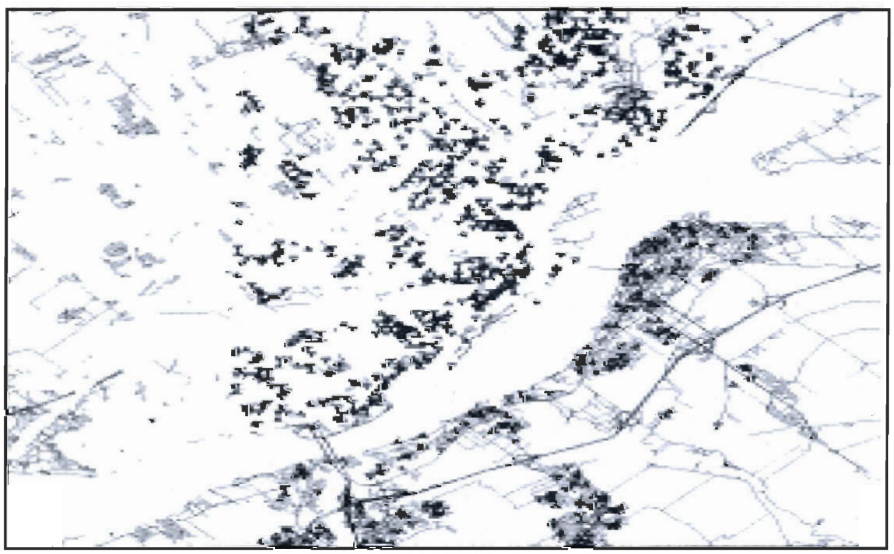


Figure 5.6 Revenus des ménages avec deux travailleurs de sexes opposés

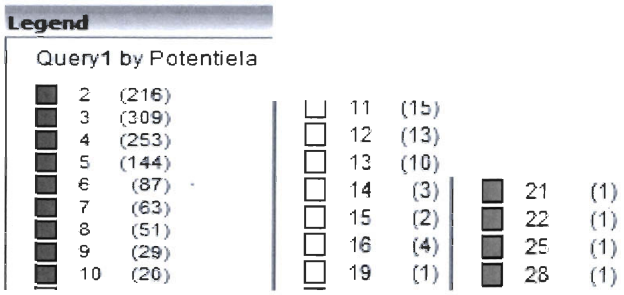


Figure 5.7 Légende des codes de couleurs de la figue 5.6

5.3.3.2 Ménage avec un seul travailleur

Nous distinguons deux cas : un ménage avec un travailleur de sexe féminin ou un ménage avec un travailleur de sexe masculin. Nous allons analyser chaque cas séparément et nous allons présenter toutes les étapes de notre démarche.

Si la personne est de sexe masculin, c'est à dire le champ sexe est égale à 1. On extrait les ménages ayant des travailleurs de ce sexe via la requête suivante :

```
SELECT uniqueMena, uniquePers, prof_type, sexe, age,
occupation
FROM personnes01WHERE sexe=1 And occupation=1;
```

- On cherche leurs occupations et type de profession avec cette requête :

```
SELECT ProfSexe1.uniqueMena, ProfSexe1.UniquePers,
ProfSexe1.sexe, ProfSexe1.age, ProfSexe1.prof_type,
ProfSexe1.occupation FROM ProfSexe1, rofFBiParental  WHERE
rofSexe1.UniqueMena<>ProfFBiParental.UM1;
```

- On fait une jointure entre les tables intermédiaires précédentes et la table qui représente notre liste de revenus pour trouver les poids de chaque profession. La table *RevenuT1* qui enregistre les revenus d'un ménage dont le travailleur est un homme est obtenue par la requête suivante :

```
SELECT UniqueMena, age, sexe, prof_type, CodeRMenage,
PoidRevenuTotal FROM RevenuMenage, ProfMParental1
WHERE CodeRConjoint1=prof_type;
```

- On agrège les données pour enlever les doublons avec les requêtes :

```
SELECT UniqueMena, CodeRMenage FROM RevenuT1 GROUP BY UniqueMena,
CodeRMenage HAVING count(*)>1;
```

- On effectue une jointure sur les *uniquemena* des tables *RevenuT1,* et *ResultatDeficit* pour trouver les ménages ayant des travailleurs hommes qui appartiennent à une catégorie de déficit. Le résultat est la table *RDeficit1*

```
SELECT ResidGrHex, RevenuUnTravailleur1.UniqueMena,
ResultatDeficit.age, LPasserBus, Occupation, NbEtudMenage,
prof_type, CodeRMenage, PoidRevenuTotal, NbAuto, NBCondM,
TotalCondMenage, deficit FROM ResultatDeficit, RevenuT1 WHERE
ResultatDeficit.UniqueMena=RevenuUnTravailleur1.UniqueMena;
```

- Finalement on agrège les données en sommant les poids des revenus par cellules.

```
SELECT ResidGrHex, count (UniqueMena) AS NbMCel, sum(NbEtudMenage)
AS NBECel, sum(NbCondM) AS NbCondMCel, sum(NbAuto) AS NbAutoCel,
sum(TotalCondMenage) AS TotalCondMCel, sum(Deficit) AS deficitCel,
sum(CodeRMenage) AS PotentielAchat
FROM RDeficit1 GROUP BY (ResidGrHex)
HAVING count(*)>1;
```

Dans le cas ou la personne est de sexe féminin, c'est à dire le champ sexe est égal à 2. Les requêtes sont les suivantes :

```
SELECT uniqueMena, uniquePers, prof_type, sexe, age,
occupation
FROM personnes01 WHERE sexe=2 And occupation=1;
```

- On détermine leurs occupations et leur type de profession avec cette requête :

```
SELECT ProfSexe2.uniqueMena, ProfSexe2.UniquePers,
ProfSexe2.sexe, ProfSexe2.age, ProfSexe2.prof_type,
ProfSexe2.occupation
FROM ProfSexe2, ProfFBiParental WHERE
ProfSexe2.UniqueMena<>ProfFBiParental.UM2;
```

- La table *RevenuT2* qui enregistre les revenus d'un ménage dont le travailleur est une femme est obtenue par la requête de jointure suivante :

```
SELECT UniqueMena, age, sexe, prof_type, CodeRMenage,
PoidRevenuTotal FROM RevenuMenage, ProfMParental2 WHERE
CodeRConjoint1=prof_type;
```

- À l'image de l'opération effectuée pour la table *RevenuT2*, on utilise la même requête pour enlever les doublons.

```
SELECT UniqueMena, CodeRMenage FROM RevenuT2 GROUP BY
UniqueMena, CodeRMenage HAVING count (*)>1;
```

- Pour les tables *RevenuT2* et *ResultatDeficit,* on effectue une jointure sur les *Uniquemena* pour trouver les ménages ayant des travailleurs de sexe féminin qui appartiennent à une catégorie de déficit. Le résultat est la table *RDeficit2*.

```
SELECT ResidGrHex, RevenuUnTravailleur1.UniqueMena,
ResultatDeficit.age, LPasserBus, Occupation, NbEtudMenage,
prof_type, CodeRMenage, PoidRevenuTotal, NbAuto, NBCondM,
TotalCondMenage, deficit FROM ResultatDeficit, RevenuT2 WHERE
ResultatDeficit.UniqueMena=RevenuUnTravailleur1.UniqueMena;
```

- Enfin, nous agrégeons les données en additionnant les poids des revenus par cellules.

```
SELECT ResidGrHex, count (UniqueMena) AS NbMCel,
sum(NbEtudMenage) AS NBECel, sum(NbCondM) AS NbCondMCel,
sum(NbAuto) AS NbAutoCel, sum(TotalCondMenage) AS
TotalCondMCel, sum(Deficit) AS deficitCel, sum(CodeRMenage) AS
PotentielAchat
FROM RDeficit2 GROUP BY (ResidGrHex)
HAVING count(*)>1;
```

5.3.3.3 Conclusion

À la suite des étapes que nous avons présentées dans la section précédente, on obtient les tables *PotentielAchat1* et *potentielAchat2*. Elles expriment les revenus des ménages qui ont un déficit de voiture et qui possèdent un seul travailleur masculin ou féminin, ou qui possèdent deux travailleurs de sexe opposés. Elles sont exportées dans *Mapinfo* pour trouver les cartes suivantes :

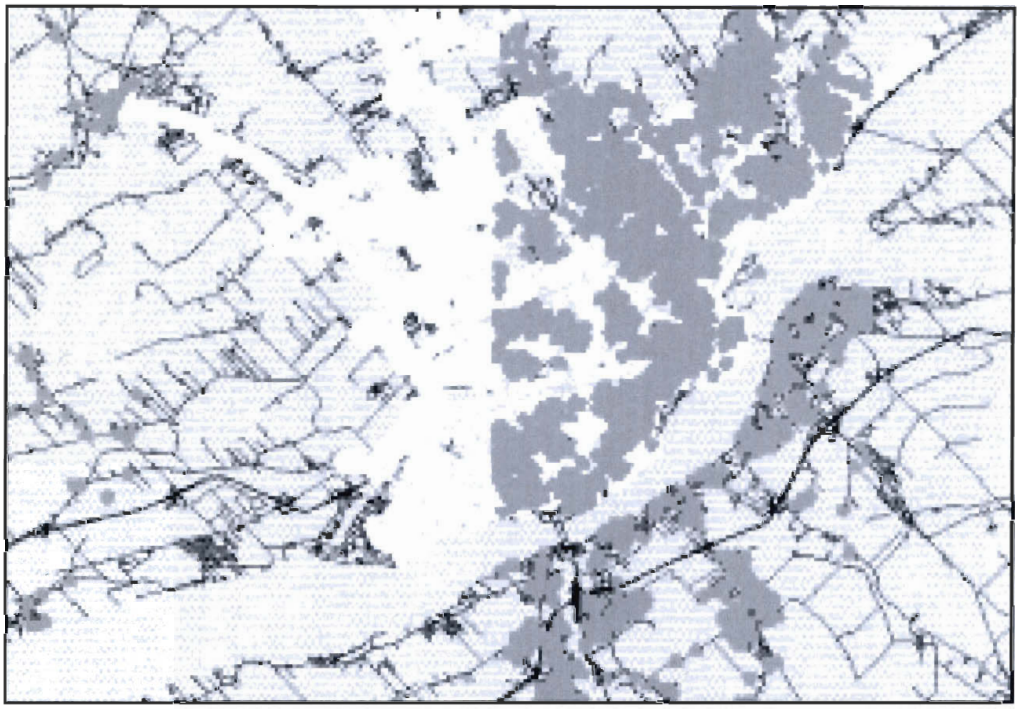


Figure 5.8 Carte de distribution des ménages avec un travailleur masculin

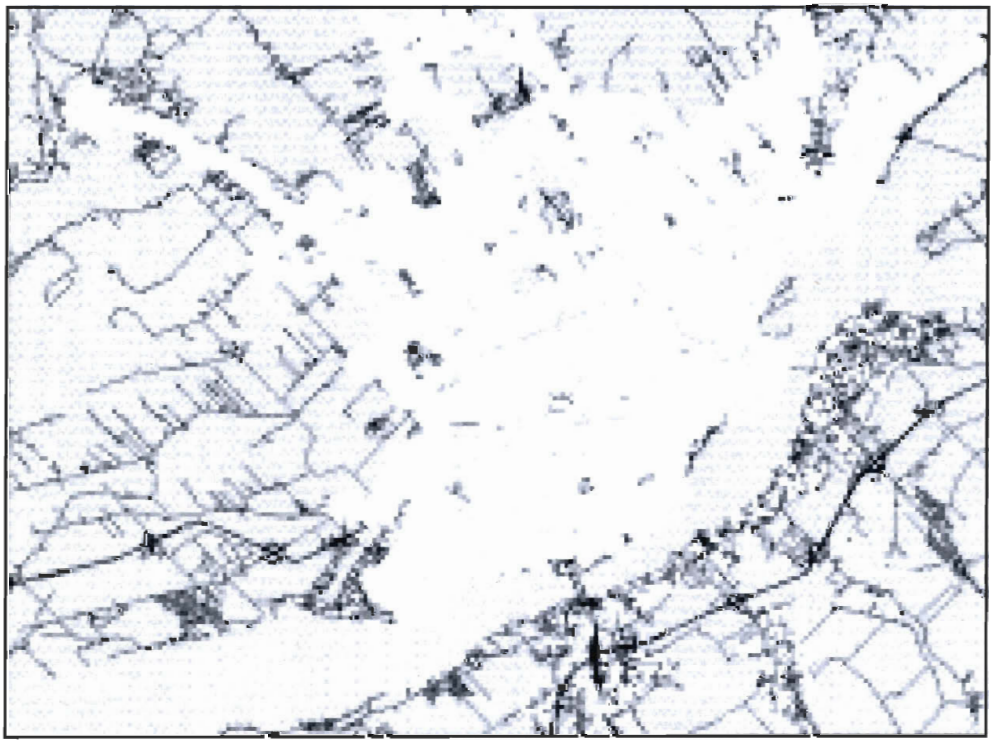


Figure 5.9 Carte de distribution des ménages avec un travailleur féminin

5.3.4 Conclusion

Nous avons analysé les données issues de la population synthétique. Nous avons localisé les cellules en déficit de voitures et nous avons établi une liste de revenus pour déterminer le revenu par ménage. Nous avons obtenu les ménages ayant un déficit de voitures et leurs revenus et par conséquent, les cellules qui peuvent potentiellement acquérir un véhicule. Cependant, toute cette analyse est relative à l'année d'étude 2001. Dans ce qui suit, nous allons présenter notre démarche pour déterminer l'évolution de notre population d'intérêt au cours des 5 années qui suivent l'année d'étude, c'est-à-dire de 2001 à 2006.

5.4 Vieillissement de la population

5.4.1 Introduction et contexte

Le but de ce procédé est de déterminer l'évolution d'une partie de la population synthétique au cours du temps. Cette étude vise à nous permettre de comparer l'évolution du potentiel d'achat en véhicules pour les années qui suivent l'année de l'étude initiale (2001). On va appliquer cette évolution pour les cinq années qui suivent, soit entre 2001 et 2006 inclusivement.

Ainsi, nous chercherons à évaluer le devenir des zones génératrices de véhicules entre 2001 et 2006. On pose les hypothèses suivantes :

- La taille de population est stable. C'est-à-dire, on ne considère pas l'évolution démographique (naissances et décès).
- La migration d'un ménage vers un autre domicile, ne va pas influencer l'achat d'un véhicule.
- L'âge moyen pour commencer à aller à l'université est 20 ans
- Les adolescents vont acquérir leur permis à 16 ans.

De plus, nous ne nous intéressons qu'au vieillissement des adolescents au cours de ces années, car ce sont des personnes qui peuvent acquérir un permis de conduire au cours des années d'études.

Cependant, le problème qui se pose est le fait que la population synthétique ne nous fournit que des catégories d'âges et non les véritables âges des enfants. Pour résoudre ce problème,

nous proposons d'assigner un âge aléatoire à chaque enfant à l'intérieur des catégories d'âge 10-14 ans et 15-19 ans, car ce sont les tranches d'âges des conducteurs potentiels. Ensuite pour chaque année, à partir de 2001 on rajouter 1 à ces âges aléatoires afin de trouver s'ils appartiendront ou non à la catégorie des conducteurs potentiels, c'est-à-dire 16 ans et plus. L'assignation aléatoire des âges aux enfants d'un ménage ne se fera qu'une seule fois pour la population de l'année2001. Ensuite on rajoute la valeur 1 pour faire le vieillissement de chaque enfant pour les années qui suivent.

5.4.2 Démarche de vieillissement de la population

Étant donné que chaque personne dans la table *personnes01* appartient à une catégorie d'âge et ne possède pas un âge en tant que tel, il fallait assigner aléatoirement une valeur qui représente un âge dans cette catégorie. Notons, qu'on s'intéresse seulement aux catégories d'âge 3 (10-14 ans) et 4 (15-19 ans) (voir annexe A), c'est-à-dire les enfants qui peuvent accéder à un permis de conduire dans les prochaines six années, et par conséquent devenir des conducteurs potentiels. On ne s'intéresse pas au vieillissement des autres personnes de la population (en particulier les adultes et jeunes enfants) car nous supposons que ce vieillissement n'influence pas l'acquisition d'un nouveau permis de conduire. Pour faire vieillir les jeunes personnes des catégories 3 et 4, nous avons adopté la démarche suivante :

a) Programme de génération de chiffres aléatoire

Premièrement, nous avons généré des âges aléatoires. Nous avons utilisé un code C# dans un environnement .NET 2005. (Voir annexe A.2). Le code est simple, il se connecte à la base de données, et il insère des valeurs générées aléatoirement entre l'intervalle *(10,14)* et l'intervalle *(15,19)*. Remarquons que le nombre d'itérations de la génération aléatoire pour chaque intervalle, correspond respectivement au nombre d'enregistrements des personnes appartenant aux catégories 3 et 4, qui sont déterminés par des requêtes SQL.

b) Modèle conceptuel de la base de données :

Deuxièmement, nous avons adopté le modèle conceptuel de la base de données. En effet, une première approche consistait à ajouter un autre champ d'âge à la table *personnes01*.

Toutefois, ce champ est le seul qui variera pour les 5 années futures, tandis que les autres champs resteront constants. Par conséquent, il y aura de la redondance des données. Pour résoudre ce problème, nous avons rajouté une autre table : *AgePersonnesJeunes01,* qui a comme attributs: la clé *UniquePers* de la table *personnes01* et *l'age* qui est généré aléatoirement par notre code. Par exemple :

AgePersonnesJeunes01	
UniquePers	AgeP
40	12
42	14
53	14

Tableau 5-9 Age d'un conducteur potentiel en 2001

Ainsi, grâce à la clé étrangère *UniquePers*, on peut retracer les autres informations relatives à une personne. Finalement, nous avons structuré les autres tables des années subséquentes de la même manière. Il ne reste plus qu'à établir le lien entre ces différentes tables. Nous avons eu deux idées : la première consistait à chaîner les tables l'une à la suite de l'autre via une clé étrangère. Toutefois, ceci complexifie les requêtes, puisque pour chercher l'information d'une année donnée, il faut trouver la clé étrangère des tables qui la précèdent (recherche en cascade). La deuxième idée que nous avons retenue, est d'avoir un regroupement dans une seule table *CatégorieAge* ayant comme clé unique une clé composée de la clé *UniquePers,* l'âge de cette personne et l'année d'étude, car une personne ne peut avoir qu'un seul âge par année. Dans ce qui suit, nous présenterons les étapes pour peupler ces tables.

c) Peuplement des nouvelles tables

On fait une sélection sur la table *Personnes01* pour obtenir les *UniquePers* dont la catégorie d'âge est 3 ou 4 via la requête suivante :

```
Select UniquePers From Personnes01 where age = 3 or age= 4
```

On obtient une seule colonne que l'on copie dans une table *AgePersonnesJeunes01* et on insère les nombres aléatoires dans cette dernière. Ensuite, pour les années qui suivent, on crée une copie de cette table (copier/coller) et on effectue une mise à jour sur le champ *AgeP* en rajoutant 1 à l'aide de la requête de mise à jour suivante :

```
UPDATE AgePersonnesJeunes01  SET ageP = ageP+1;
```

On répète les requêtes de ce processus 5 fois pour chaque année.

```
UPDATE AgePersonnesJeunes02 SET ageP = ageP+1;
UPDATE AgePersonnesJeunes03 SET ageP = ageP+1;
UPDATE AgePersonnesJeunes04 SET ageP = ageP+1;
UPDATE AgePersonnesJeunes05 SET ageP = ageP+1;
UPDATE AgePersonnesJeunes06 SET ageP = ageP+1;
```

Un exemple du résultat obtenu est le suivant. Reconsidérons la table *AgePersonnesJeunes01* présentée plus haut, la personne identifiée par le numéro 40 a 12 ans en 2001, en 2002 : 13 ans, en 2003 : 14 ans, en 2004 : 15 ans, en 2005 16 ans (conducteur potentiel) et en 2006 : 17 ans. (Voir les tableaux suivants)

AgePersonnesJeunes02	
UniquePers	AgeP
13	19
40	13
42	15

Tableau 5.11 Age d'un conducteur potentiel en 2002

AgePersonnesJeunes03	
UniquePers	AgeP
13	20
40	14
42	16

Tableau 5.12 Age d'un conducteur potentiel en 2003

AgePersonnesJeunes04	
UniquePers	AgeP
13	21
40	15
42	17

Tableau 5.13 Age d'un conducteur potentiel en 2004

AgePersonnesJeunes05	
UniquePers	AgeP
13	22
40	16
42	18

Tableau 5.14 Age d'un conducteur potentiel en 2005

AgePersonnesJeunes06	
UniquePers	AgeP
13	23
40	17
42	19

Tableau 5.15 Age d'un conducteur potentiel en 2006

5.4.3 Analyse des données

Dans cette section, nous allons présentons notre démarche pour analyser l'évolution de la population au cours du temps, c'est-à-dire entre les années 2001 et 2006.

5.4.3.1 Données descriptives

Premièrement, nous avons trouvé les ménages auxquels appartiennent ces jeunes conducteurs, ensuite les cellules contenant ces ménages. Comme nous avons déjà identifié les personnes dans l'étape précédente dans la table *AgePersonnesJeunes01,* il suffisait de rajouter les ménages et les cellules contenant ces personnes. La requête est :

```
SELECT ResidGrHex, Personnes01.UniqueMena, UniquePers
FROM Menages01, Personnes01 WHERE Personnes01.UniqueMena= Menages01.UniqueMena
and
uniquePers in ( select UniquePers From AgePersonnesJeunes01 );
```

Le résultat est stocké dans la table *CellulesAdos01*. Cette dernière contient l'ensemble des cellules, qui ont des ménages possédant des conducteurs potentiels « adolescents » au cours du temps. Il faut répéter cette étape pour chaque année afin de rajouter les champs *UniqueMena* et *ResidGrHex* dans les tables *AgePersonnesJeunes02, AgePersonnesJeunes03, AgePersonnesJeunes04, AgePersonnesJeunes05,* et *AgePersonnesJeunes06.* Il est à remarquer que nous aurions pu faire cela avant de faire vieillir la population, c'est-à-dire avoir la table *AgePersonnesJeunes01* que nous avons présentée plus haut, avec deux champs de plus : *UniqueMena* et *ResidGrHex*, ensuite faire les mises à jour sur le champ *AgeP* (voir tableau 5.16). Ensuite, pour ces cellules et ces ménages, il fallait extraire le nombre des personnes âgées de 16 ans par cellule pour chaque année. Ce procédé est répété 5 fois pour chaque année. Les requêtes et les tables sont présentées dans l'annexe D.

5.4.3.2 Données spatiales

Une fois que les données sont préparées, il faut les intégrer dans *MapInfo* et localiser les cellules sur la carte de la ville de Québec. Notons que nous avons choisi de représenter la densité des conducteurs potentiels par des nuages de points. En effet, un point correspond à un conducteur potentiel. Cette densité est colorée en rouge sur les cartes. Les

lignes en vert représentent les lignes d'autobus. Les requêtes spatiales utilisées et les cartes obtenues sont les suivantes :

```
Selecet * from NbAdos02, Grille_Triangulaire
Where GrilleTriangulaire.Cellid= NbAdos02.ResidGrHex
```

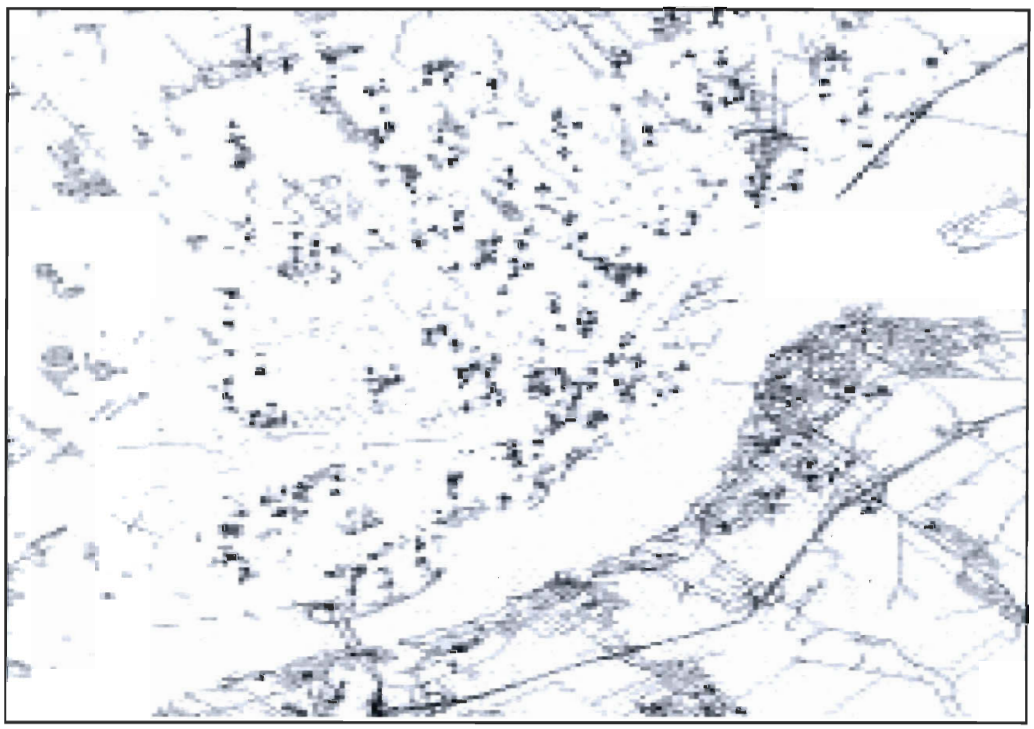


Figure 5.10 Carte de densité des conducteurs potentiels en 2002

```
Selecet * from NbAdos03, Grille_Triangulaire
Where GrilleTriangulaire.Cellid= NbAdos03.ResidGrHex
```

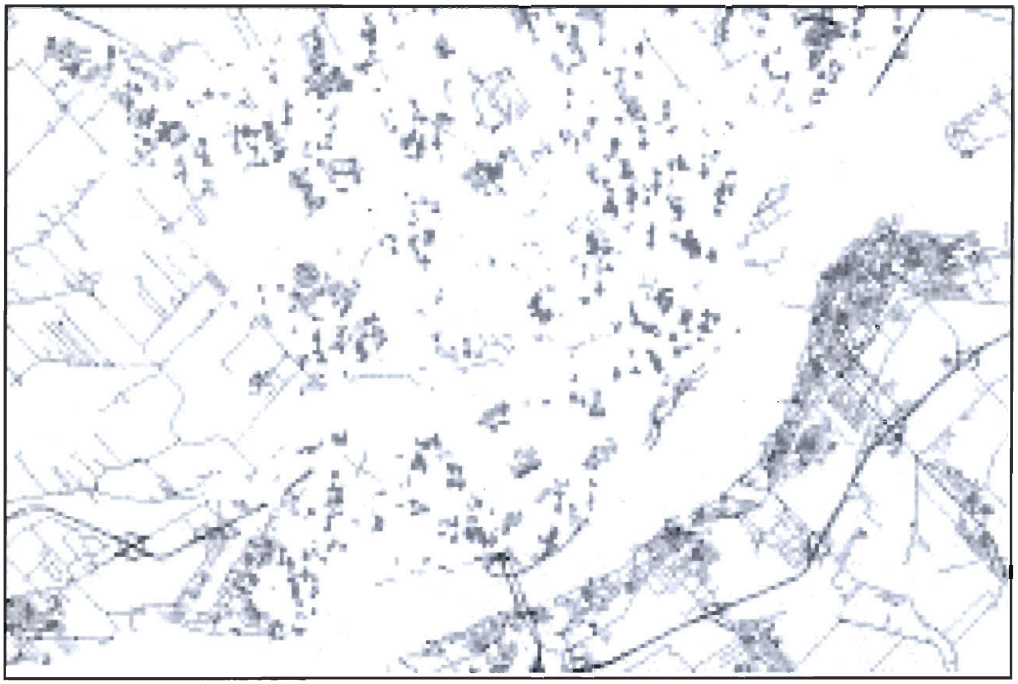


Figure 5.11 Carte de densité des conducteurs potentiels en 2003

```
Selecet * from NbAdos04, Grille_Triangulaire
Where GrilleTriangulaire.Cellid= NbAdos04.ResidGrHex
```

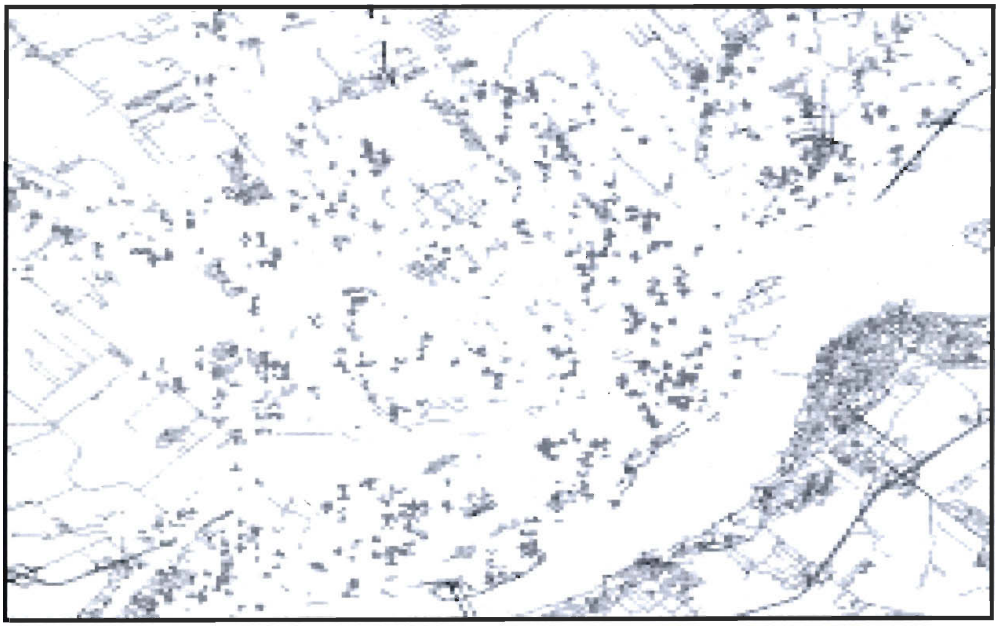


Figure 5.12 Carte de densité des conducteurs potentiels en 2004

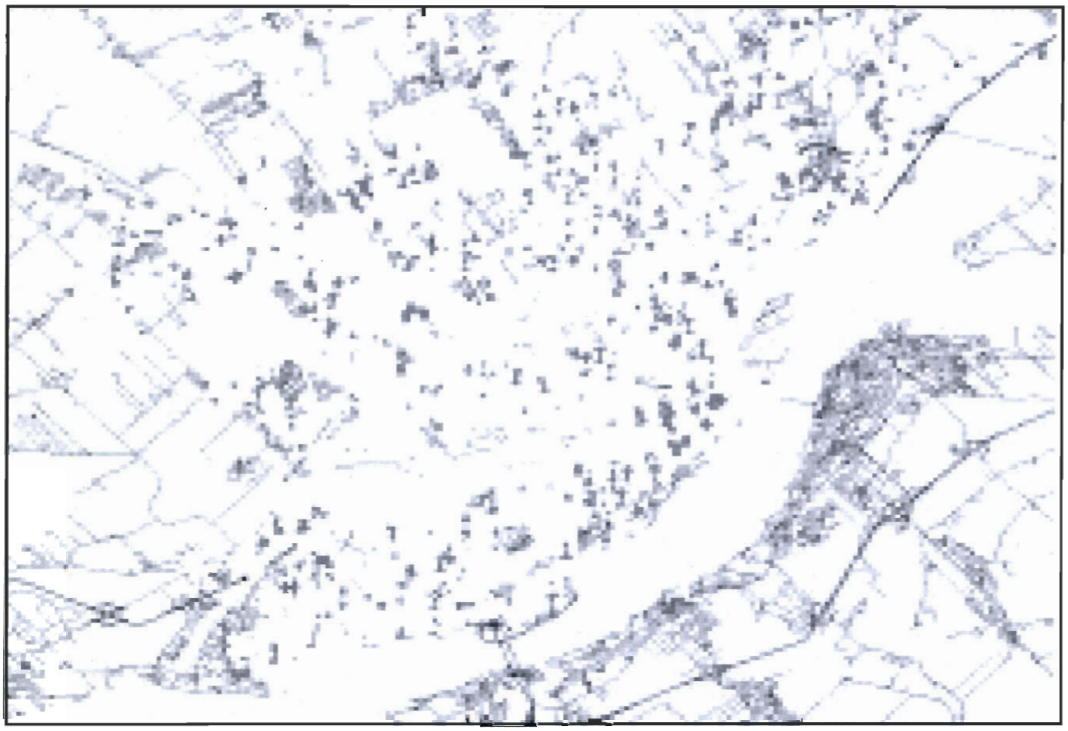


Figure 5.13 Figure 5.13a Carte de densité des conducteurs potentiels en 2005

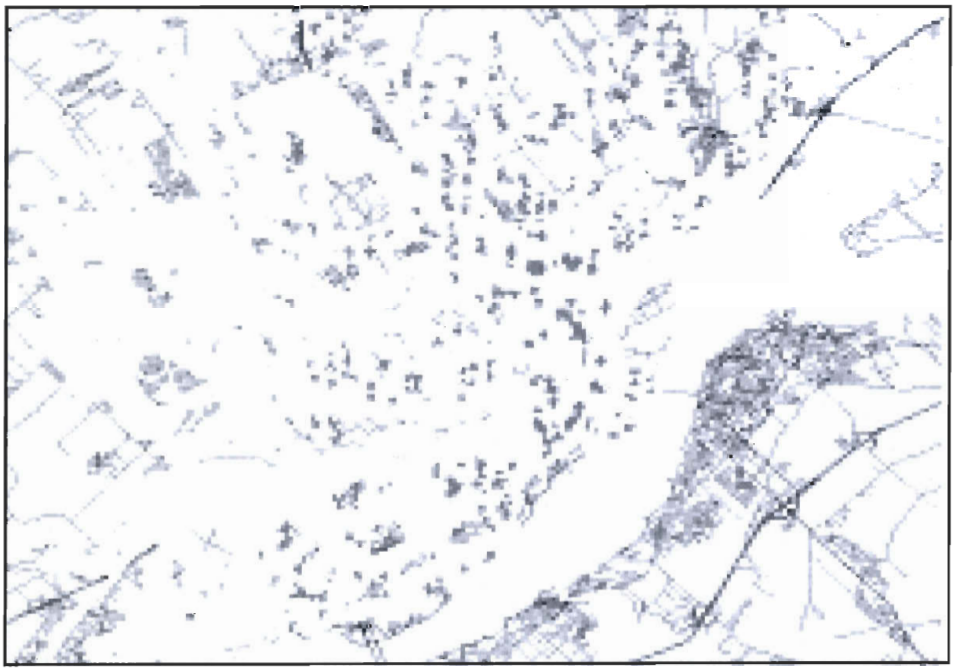


Figure 5.14 Carte de densité des conducteurs potentiels en 2006

Selecet * from NbAdos06, Grille_Triangulaire Where GrilleTriangulaire.Cellid= NbAdos06.ResidGrHex

5.5 Conclusion

Dans ce chapitre, nous avons établi un processus d'analyse spatiale afin d'extraire des données plus fines à partir des données de la population synthétique. Premièrement, nous avons appliqué le processus d'extraction et de nettoyage des données afin de cerner celles qui nous intéressent. Par exemple, nous avons trouvé les déficits en voitures par ménage ainsi que leurs références spatiales. Autrement dit, on peut localiser chaque ménage en déficit de véhicule sur la carte de la ville de Québec. Deuxièmement, nous avons réalisé une liste qualitative pour cerner le revenu par ménage. Ensuite, à partir de ces données, nous avons établi des cartes de densités pour visualiser la population sur la carte de la ville de Québec. Ceci nous a donné une première idée sur les zones génératrices de déficits de véhicules. Nous avons remarqué, que ces zones se trouvent sur la périphérie de la ville. Finalement, nous avons fait vieillir les conducteurs potentiels de 2001 à 2006 afin d'estimer la répartition et l'impact des jeunes conducteurs sur ces zones et étudier leurs évolutions. Nous avons constaté par observation visuelle des regroupements de cellules ayant des déficits forts ou moyens. C'est exactement ce que nous cherchons à réaliser avec notre prototype que nous allons présenter dans le chapitre 6.

Chapitre 6 : Architecture et approches de clustering

6.1 Introduction

Comme indiqué dans le chapitre 5, nous avons localisé des zones qui possèdent des ménages en déficit de véhicules et qui peuvent en acquérir un ou plusieurs en fonction de leurs revenus et du nombre de leurs conducteurs potentiels. Cependant, nous avons dû les interpréter par simple observation. Dans ce chapitre nous allons développer un outil d'analyse spatiale qui permettrait d'identifier automatiquement ces zones génératrices d'excès de véhicules et de les localiser sur la carte. De plus, les grilles que nous avons présentées dans le chapitre 4 ont des propriétés à la fois géométriques et sémantiques. Elles sont géométriques puisque chaque cellule est géo-référencée et a une forme régulière. Elles sont sémantiques car on peut informer notre environnement à partir des données de la population que nous avons présentées dans le chapitre 5. Ainsi, on parlera d'un environnement « informé ». Dans ce qui suit, nous présentons notre outil d'un point de vue de génie logiciel, c'est-à-dire le modèle de classe, les technologies que nous avons utilisées pour le développer, son architecture et l'approche que nous avons retenue pour faire les regroupements. Finalement nous présentons les résultats obtenus.

6.2 Développement et architecture du système

Dans cette section, nous exposons les détails techniques de notre système. Premièrement, nous présentons l'environnement technique que nous avons utilisé. Ensuite, nous détaillons l'architecture du système et finalement, son modèle conceptuel.

6.2.1 Technologie et processus de développement

Dans cette section nous présentons les motifs de nos choix technologiques au sujet de l'environnement technique et du processus de développement.

6.2.1.1 Environnement technique

Nous avons implémenté notre prototype avec la technologie « .NET » et plus particulièrement avec le langage de développement C#. Les motifs de ce choix sont les suivants :

- Interopérabilité avec les solutions existantes au sein du GRIC.
- Notre maîtrise de cet environnement.
- Interopérabilité avec la librairie Piccolo.NET pour la gestion des opérations graphiques.
- La robustesse de la plateforme.

Également, nous avons choisi le système de gestion de bases de données Access à cause de sa comptabilité avec le SIG MapInfo. Ainsi, nous pouvons directement transférer les données issues de l'analyse spatiale que nous avons élaborée dans le chapitre 5 vers des tables de données Access. Par conséquent, on pourrait alimenter notre prototype par ces données. Finalement, nous avons eu recours au formalisme UML pour concevoir et modéliser notre logiciel.

6.2.1.2 Processus de développement

Nous avons employé une approche par prototypage comme processus de développement. Le principe de cette approche consiste essentiellement à analyser et implanter les composantes d'une application rapidement selon des besoins qui reflètent une partie du système final. Ce genre de prototypes est appelé « prototype évolutif ». Au cours du développement, il existe plusieurs prototypes et chaque prototype à lui seul, fait l'objet d'un cycle d'analyse, de conception, d'implantation et de test afin de former un sous-ensemble du produit logiciel final. Ce dernier est alors raffiné de façon incrémentale, jusqu'à obtenir le produit final. Ce type d'approche ne doit pas être confondu avec l'approche de prototypage jetable qui consiste à concevoir rapidement une application, sans soin particulier, pour qu'elle soit écartée une fois qu'elle aurait rempli son rôle. Généralement, on s'en sert pour aider à clarifier ou à formuler les besoins. Elle consiste à réaliser rapidement et dès le début du cycle de vie un prototype de l'application qui va permettre de valider les spécifications. Pour conclure, le prototypage est une approche appropriée dans certains contextes exploratoires.

6.2.2 Architecture et conception du système

Nous allons présenter dans cette section, une vue d'ensemble des modules de notre application. Ensuite, nous présentons un diagramme de composants afin de montrer les associations existantes entre ces modules. Finalement, pour chaque module, nous

présentons une vue plus détaillée de ses classes et nous expliquons succinctement nos choix de conceptions en ayant recours aux designs patterns comme méthode de modélisation.

6.2.2.1 Architecture du système

La figure suivante montre une vue globale des différent packages qui sont développés ainsi que le processus d'injection des données dans notre système. On distingue trois grandes couches :

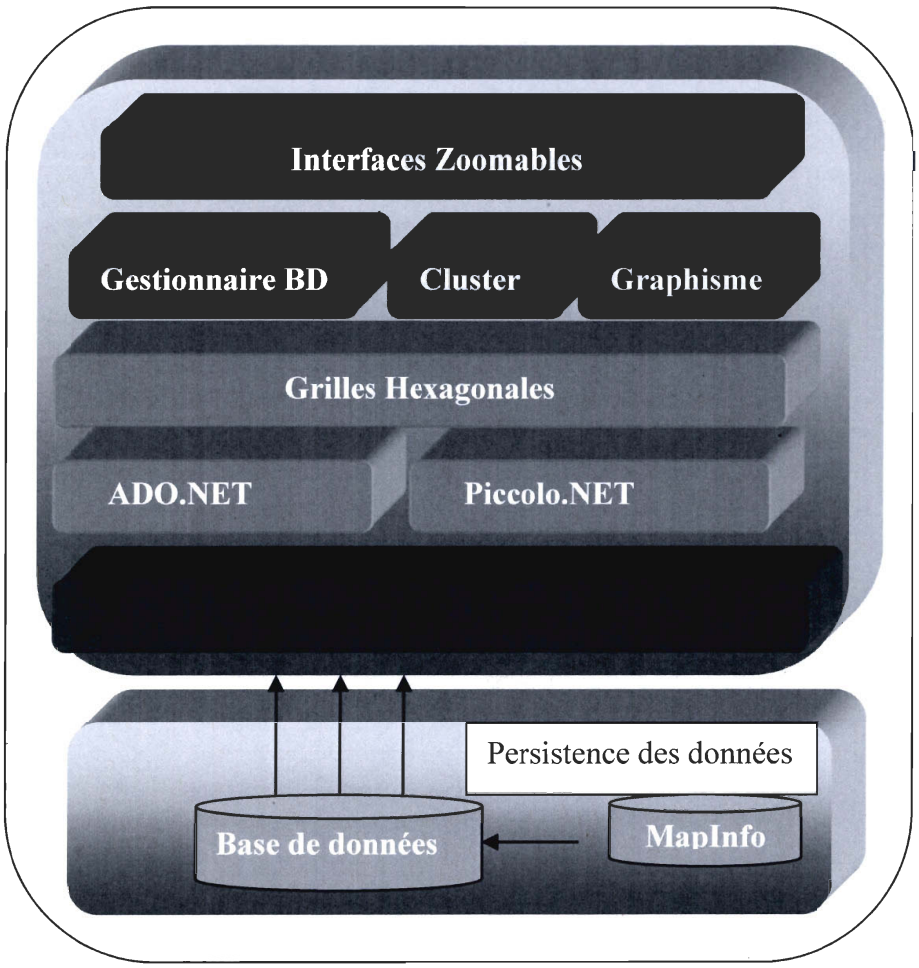


Figure 6.1 Architecture du système

- La couche de persistance des données : Enregistre les données dans des bases de données géographiques. Techniquement ceci est assuré par le framework *.NET* et en particulier, sa librairie *ADO.NET*. Les données sont extraites du SIG MapInfo et transférées à la base de données sous un format Access. C'est cette dernière qui

alimentera notre système avec les mêmes données que nous avons utilisées dans le chapitre 5.

- La couche métier ou logicielle : Responsable du fonctionnement du système, elle se compose des packages suivants :
  - **Gestionnaire BD :** Il gère toutes les opérations avec les bases de données comme charger, sauvegarder et mettre à jour les données.
  - **Cluster :** Il assure une approche de regroupement selon les techniques que nous avons présentées dans la section data mining spatial du chapitre 2.
  - **Graphisme :** Il est responsable de toutes les opérations d'affichage en se basant sur les routines de la librairie Piccolo.NET
  - **Grilles hexagonales :** Il construit et structure les grilles hexagonales que nous avons présentées dans le chapitre 4.
- La couche présentation : C'est l'interface usager qui représente une façade via laquelle l'usager interagira avec l'application. Comme nous l'avons précisé dans le chapitre 2, ces interfaces ont la particularité d'être des interfaces zoomables (ZUI). Un guide usager est présenté en annexe (voir annexe C) pour montrer le fonctionnement des interfaces du système. Elle se compose des fenêtres suivantes :
  - **Afficheur :** Fenêtre qui permet de voir les résultats sur une carte
  - **Outil de manipulation des données :** Fenêtre qui permet à l'usager de manipuler les données sémantiques.
  - **Partitionnement des grilles :** Permettra de créer des grilles paramétrées selon les attributs que nous avons mentionnés dans le chapitre 4.
  - **Forme Principale :** La fenêtre principale qui contient toutes les fenêtres précédentes. Elle contient une barre d'outil et des menus pour accéder aux opérations.

Dans ce qui suit on présente le diagramme de composants afin de montrer les relations entre ces différents modules.

Le module *Graphisme* contient les routines nécessaires pour les opérations d'affichage. Il fonctionne en étroite relation avec la bibliothèque *Piccolo.NET*. Le module *Inteface ZUI*, dépend du module responsable de la création des grilles. À son tour, ce dernier a besoin du package *GestionnaireBD* pour les opérations de prétraitement. Finalement, le package

Cluster utilise le module *Grille hexagonale* pour appliquer les méthodes de regroupement et de visualisation au niveau du module *Interfaces ZUI*. La figure 6.2 montre le diagramme de composants.

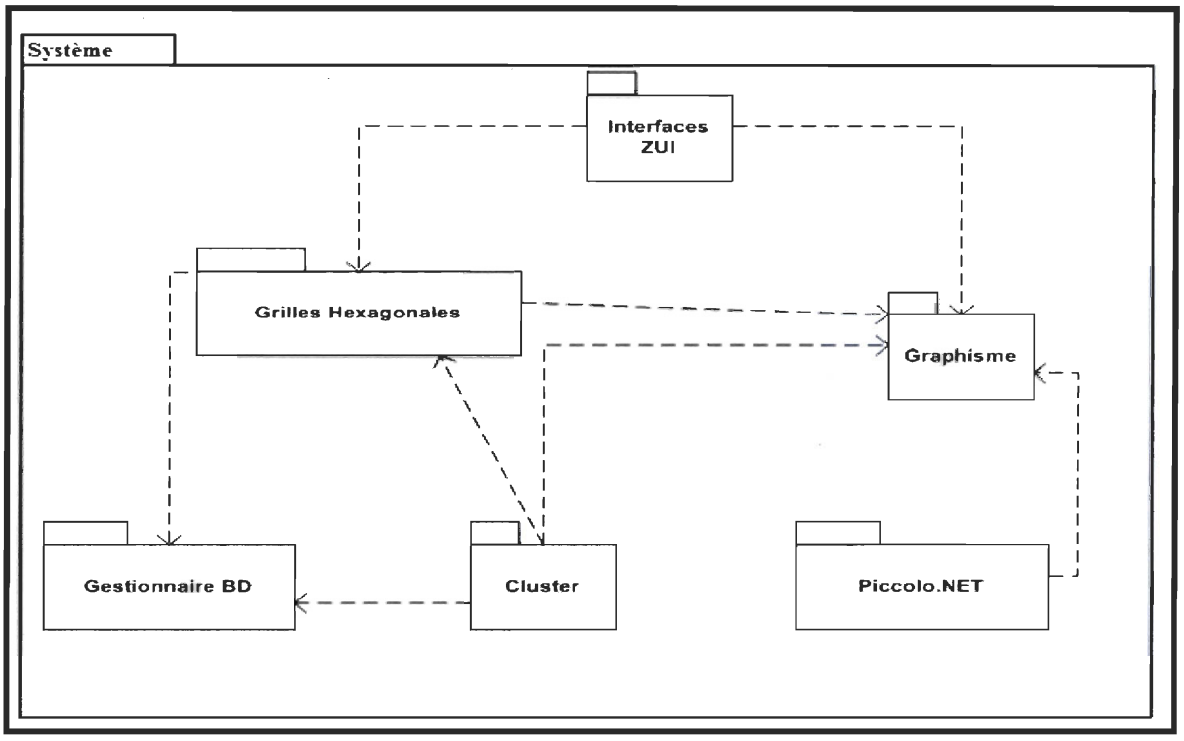


Figure 6.2 Diagramme de composants du système

Dans cette section nous avons présenté l'architecture de notre système ainsi que les différents packages qui la constituent. Par ailleurs, nous avons présenté brièvement les fonctionnalités de chaque package et le diagramme de composants qui montre leurs interactions. Dans ce qui suit, nous abordons la conception de cette architecture et ses composantes de manière plus détaillée.

6.2.2.2 Conception du système

Pour chaque module indiqué plus haut, nous présentons ses classes dans la section suivante. Nous allons premièrement présenter les patrons de conception (design patterns), puisque nous avons eu recours à cette technique lors de la conception du système. Ensuite nous montrerons les classes utilisées. Rappelons que l'annexe B.2 présente le dictionnaire des classes du système.

6.2.2.2.1 Patrons de conception

Dans ce qui suit, nous présentons quelques généralités sur les patrons de conception, la manière dont ils sont organisés, leurs avantages et leurs inconvénients.

a) Définition :

D'abord, rappelons des notions de base sur les designs patterns « *GOF* ». Ces derniers sont des micro-architectures reconnues pour leur utilité dans le développement logiciel. En effet, ils aident le développeur des deux manières suivantes : ils offrent des solutions qui facilitent la maintenance et la réutilisation du code. Ce sont des solutions flexibles apportées à des problèmes fréquents.

b) Catalogue des patrons et organisation :

Un pattern est formé par un nom, un problème et une solution. Il affecte des rôles aux classes qui participent aux objectifs de ce pattern. Le tableau 6.1 catégorise ces patterns selon leurs rôles et leurs domaines. Par domaine, on sous-entend la nature du pattern en tant que telle. Le tableau est organisé selon deux critères : le rôle et le domaine.

		Rôles		
		Créateur	**Structurel**	**Comportement**
Domaine	**classe**	**Fabrication**	**Adapter**	**Interprète**
				template
	Objet	**abstract factory**	**Adapter (objet)**	**CoR**
		Builder	**proxy**	**command**
		Prototype	**composite**	**Iterator**
		singleton	**decorater**	**Observator**
			façade	**state**
			flywight	**startegie**
			procuration	**visitor**

Tableau 6-1 Catalogue des designs patterns

Le rôle : il définit ce que fait un pattern. Il peut avoir un rôle structurel, créateur ou comportemental. Les patterns créateurs sont responsables de la création d'objets. Les modèles structuraux s'occupent de la composition de classes ou d'objets. Tandis que les patrons comportementaux spécifient les interactions entre classes et répartissent les responsabilités.

Le domaine : Il précise si le modèle s'applique à des classes ou à des objets. Ceux qui s'appliquent aux classes, traitent les relations entre les classes et leurs sous-classes. Ces relations sont établies par héritage : elles sont donc statiques, c'est-à-dire établies lors de la compilation ou interprétation. Par contre, les patterns qui sont assujettis aux objets, gèrent les relations entre objets, ils sont dynamiques. Les modèles créateurs de classes délèguent une partie de la création d'objets à des sous-classes, tandis que ceux qui sont créateurs d'objets délèguent cette tâche à d'autres objets. Les modèles structuraux de classes composent les classes par le biais de l'héritage alors que les modèles d'objets décrivent les façons d'assembler les objets. Quant aux modèles comportementaux de classes, ils utilisent l'héritage pour décrire les algorithmes et les flux de contrôle.

c) Avantages des patrons de conception:

La solution qui maximise la réutilisation réside dans l'anticipation des nouvelles fonctionnalités et des besoins de modifications de celles existantes, mais aussi d'une manière de concevoir un système qui facilite l'évolution dictée par cette anticipation. Pour ce faire, il faut être conscient qu'un système change fréquemment lors de son cycle de vie. Ces modifications peuvent être l'ajout de classes, modifications des clients, nouveaux tests. À la suite d'un changement, l'architecture du système est affectée et la nouvelle conception peut s'avérer coûteuse. Les design patterns évitent cela et garantissent que le système évolue et varie indépendamment des autres composantes. Nous allons exposer quelques problèmes coûteux à la conception.

- *Créer un objet en spécifiant explicitement sa classe* : cela implique une implémentation spécifique qui peut compliquer les modifications futures. Pour éviter cela, il faut créer des objets indirectement via les patrons *factory, abstract factory* ou *prototype.*
- *Assujettissement à la représentation d'un objet ou son code :* Lorsqu'un objet change, sa représentation interne peut être modifiée. En cachant cette information aux clients, on coupe court aux modifications en cascade. Cela se fait grâce aux patrons *momento, pont, procuration* et *fabrique abstraite*.
- *Assujettissement à un algorithme :* Les objets qui dépendent d'un algorithme doivent changer si l'algorithme change. Dans ce cas, il faut isoler ces derniers. Cela

se fait grâce aux modèles *monteur, itérateur, stratégie, patron de méthode* et *visiteur.*

- ❖ *Extension des fonctionnalités par sous-classes :* La dérivation d'une nouvelle classe apporte une plus grande compréhension de la classe parente. Ainsi, surcharger une opération peut imposer d'en surcharger une autre. Surtout, les sous-classes peuvent engendrer une prolifération de classes, car on peut introduire plusieurs sous-classes pour plusieurs fonctionnalités nouvelles.

Les design patterns représentent une solution souple aux quelques problèmes de conception cités précédemment, mais comment peut-on choisir un bon patron de conception ? Il faut :

- Prendre en considération la manière dont les modèles de conception résolvent les problèmes de conception.
- Explorer les intentions de chaque pattern.
- Étudier les interactions entre les patterns

Les design patterns présentent quelques désavantages. Excepté leur abstraction et la pertinence du choix d'un patron à un problème donné, ils peuvent avoir quelques effets nocifs lors de leur intégration au sein d'un système, car ils doivent être souvent adaptés selon l'intention de leurs utilisations. Donc il peut y avoir un éparpillement dans le code, ainsi qu'un mélange avec des classes existantes. De plus, si plusieurs design patterns sont utilisés dans un système, il peut devenir difficile de tracer une instance particulière.

d) Conclusion :

Nous avons présenté les patrons de conception afin de mieux comprendre notre utilisation de cette méthode au niveau de la conception logicielle de notre application. La section prochaine aborde cette étape.

6.2.2.2.2 Diagramme de classes

Tout d'abord, nous allons exposer une vue partielle du modèle des classes, c'est-à-dire pour chacun des modules de la section « Architecture système », nous présenterons ses classes et nous aborderons nos choix de conception en tenant compte des designs pattern. Finalement, nous présenterons le modèle de classes pour l'ensemble du système.

Nous nous intéressons particulièrement aux packages *grilles hexagonales*, *cluster*, *GestionnaireBD* et *Graphisme*. Pour le module *Interfaces ZUI*, le guide utilisateur est en annexe (voire annexe C).

❖ *Grilles hexagonales :*

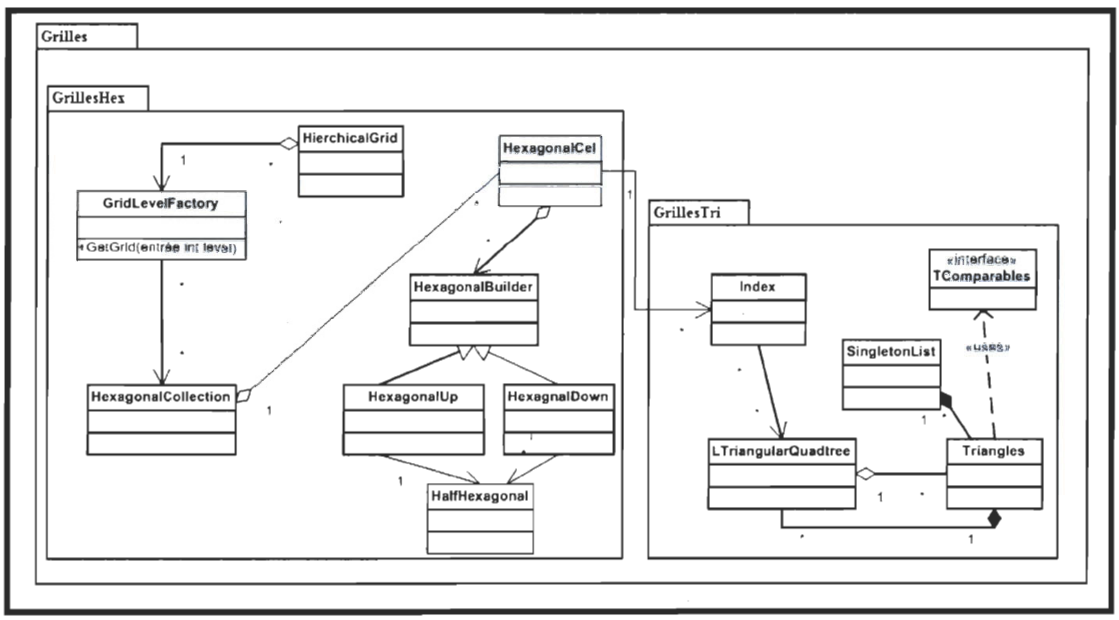


Figure 6.3 Diagramme de classe du module GrillesHexagonales

Le module *Grilles* est formé par deux autres modules qui sont *GrillesHex* et *GrillesTri*. *GrillesHex* assure la construction, le prétraitement et l'ajustement des cellules hexagonales dans les structures de données correspondantes. Ce sous-module est associé au package *GrillesTri*. Le rôle de ce dernier est de construire des cellules triangulaires, les maintenir dans les structures de données adéquates et d'assurer leurs liens avec les cellules hexagonales par le biais des index. La figure précédente montre le diagramme de classes de ce module. Dans ce qui suit nous présentons la conception de chaque sous-package.

❖ *GrillesTri:*

La classe Triangle est la classe la plus importante de notre modèle. En effet, toutes les opérations que nous avons citées dans le chapitre 4 s'y appliquent, comme par exemple le partitionnement d'un triangle. De plus, elle doit fournir une interface *TComparable* qui permettra de comparer des triangles selon leurs tailles et adjacences. C'est via cette interface que nous avons pu implémenter les règles d'adjacence géométrique.

À la suite d'un partitionnement de l'espace, chaque triangle est maintenu dans une liste globale. Elle est implémentée avec le pattern *Singleton*, car nous désirons que les triangles soient accessibles partout au sein de notre système. Chaque triangle parent est découpé en des sous-triangles enfants. Ils sont enregistrés dans la liste *LTriangularQuadtree*. De plus, chaque triangle contiendra la liste de ses enfants. Finalement, une classe *Index* permet d'exercer des opérations d'indexage sur les listes. C'est via cette classe que nous pouvons avoir un accès direct aux cellules de la grille triangulaire. Une autre approche de conception, serait d'utiliser le pattern *composite* pour enregistrer les sous-traingles. Ce dernier sera associé au pattern *Itérateur* pour parcourir la structure. Ce dernier représentera l'index.

- *GrilleHex :*

Pour créer des hexagones à partir des cellules triangulaires, l'idée est de référencer directement les triangles qui sont enregistrés dans le singleton, plutôt que de dupliquer l'information et de les enregistrer de nouveau au niveau des cellules hexagonales. Bref, chaque cellule hexagonale n'est qu'une suite de 6 chiffres qui représentent les index des triangles dans le singleton. Notre préoccupation la plus importante concernait la création d'une cellule hexagonale. Dans ce qui suit, nous détaillons notre solution conceptuelle. Comme un hexagone est formé de deux parties qui sont composées de trois triangles chacune, nous n'avons pas besoin de savoir comment ces parties sont construites. En fait, le processus de construction de ces deux parties n'est pas le même est ce, à cause des index des cellules. Par conséquent, nous voulons séparer l'entité cellule hexagonale de la construction de ses moitiés à partir des index des triangles du singleton. À cette fin, nous avons utilisé le pattern *Builder*. En effet, la classe *HexagoneCel* envoie un objet index comme paramètre à la classe *HexagonalBuilder.* Ensuite, *HexagoneCel* assemble les parties qui sont construites à partir de cet index. La classe abstraite *HexagonalBuilder* possède deux implémentations qui sont les sous-classes dérivées *HexagonalDown* et *HexagonalUp*. *HalfHexagonalcel* représente la moitié d'un hexagone que nous cherchons à construire. Une fois qu'une cellule hexagonale est construite, il faut l'enregistrer dans une liste qui représentera notre grille hexagonale à un niveau donné. Cette responsabilité est affectée à la classe *HexagonalCollection.* Elle est couplée à la classe *GridLevelFactory* qui implémente

la méthode *GetGrid.* Cette dernière est responsable de la création de la grille hexagonale selon le paramètre *level* qui correspond au niveau de résolution voulu.

- *Package Cluster :*

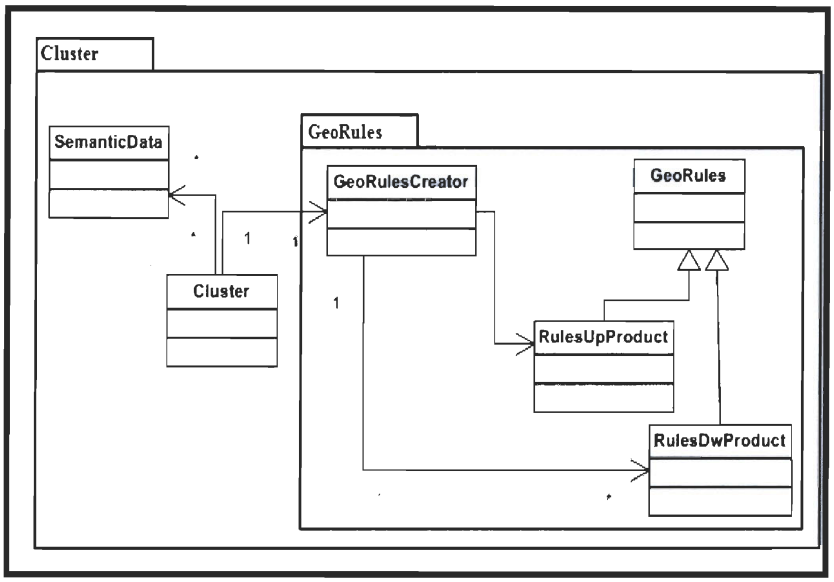


Figure 6.4 Diagramme de classe du package Cluster

Le package cluster contient le package *GeoRègles*. Ce dernier crée toutes les règles permettent de vérifier l'adjacence par côté entre deux cellules triangulaires (voir chapitre 4). En effet, comme nous spécifions une règle géométrique selon l'orientation du triangle d'intérêt, nous avons utilisé le pattern *FactoryMethod*. Ce dernier est composé par la classe *GeoRulesCreator* qui représente la *« Factory »*. Nous créons les types de règles selon l'orientation du triangle qui est passé comme paramètres à la *« Factory »*. Quant à la classe *Cluster*, elle implémente l'algorithme de regroupement (voir chapitre 2) et elle associe à chaque cellule des données sémantiques issues des bases de données. Cette classe est un lien entre les données géométriques et les données sémantiques.

- *Package Gestionnaire BD*

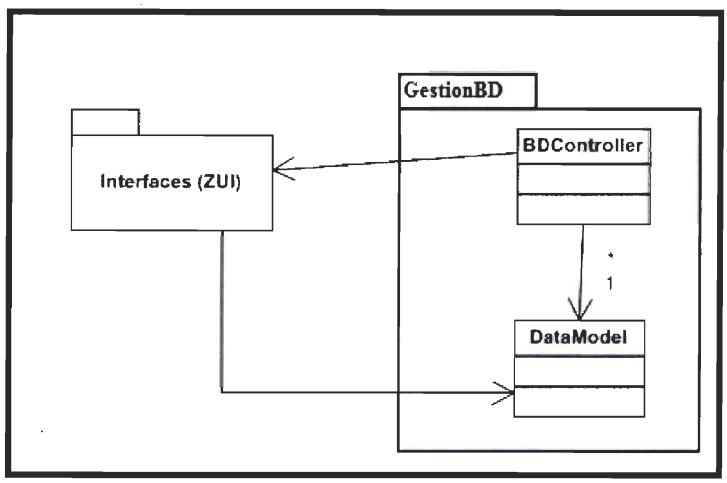


Figure 6.5 Modèle de classe du package GestionnaireBD

Dans le package *GestionnaireBD*, les classes *BDController* et *DataModel* gèrent la persistance et la matérialisation des informations au niveau des bases de données. Ce package utilise le modèle *Model-View-Controller* (*MVC*). *BDController* est le contrôleur qui reçoit les requêtes des interfaces. Il demande au modèle, dans notre cas *DataModel,* d'exécuter les traitements nécessaires. Le modèle renvoie la vue adaptée aux interfaces usagers. Il est étroitement lié au module des interfaces *Interface(ZUI).*

- *Package Graphisme*

a. :

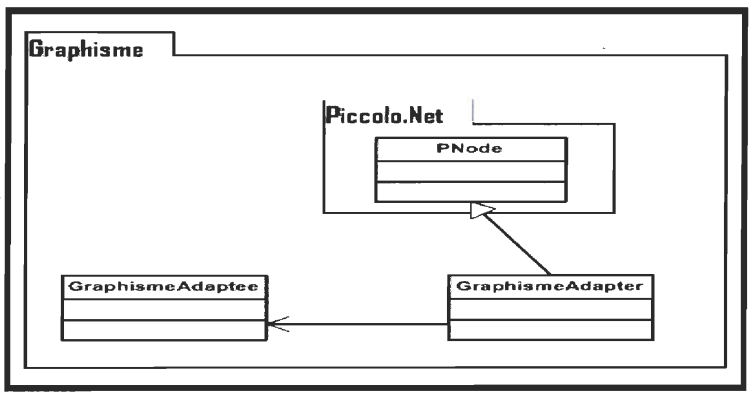


Figure 6.6 Modèle de classe du package Graphisme

Nous avons adapté les méthodes de la librairie *Piccolo.NET* à nos besoins comme pour afficher des hexagones et des triangles selon plusieurs couleurs, ou plusieurs tailles. Pour ce

faire nous avons utilisé le pattern *Adapter*. Ce dernier adapte l'interface d'une classe existante aux nouveaux besoins de la classe cliente. La classe principale dans la librairie *Piccolo.NET* est *PNode*. Nous avons fait hériter de cette classe, la classe *GraphismeAdapter* pour encapsuler les routines de la classe *PNode*. Nous avons également créée la classe *GraphismeAdaptee* dont l'interface répond à nos besoins et qui délègue les traitements de bas niveau à la classe *GraphismeAdapter*.

Finalement, la figure 6.7 montre le diagramme de classes de tout notre système. Elle présente à la fois les modules et les classes que nous avons créés.

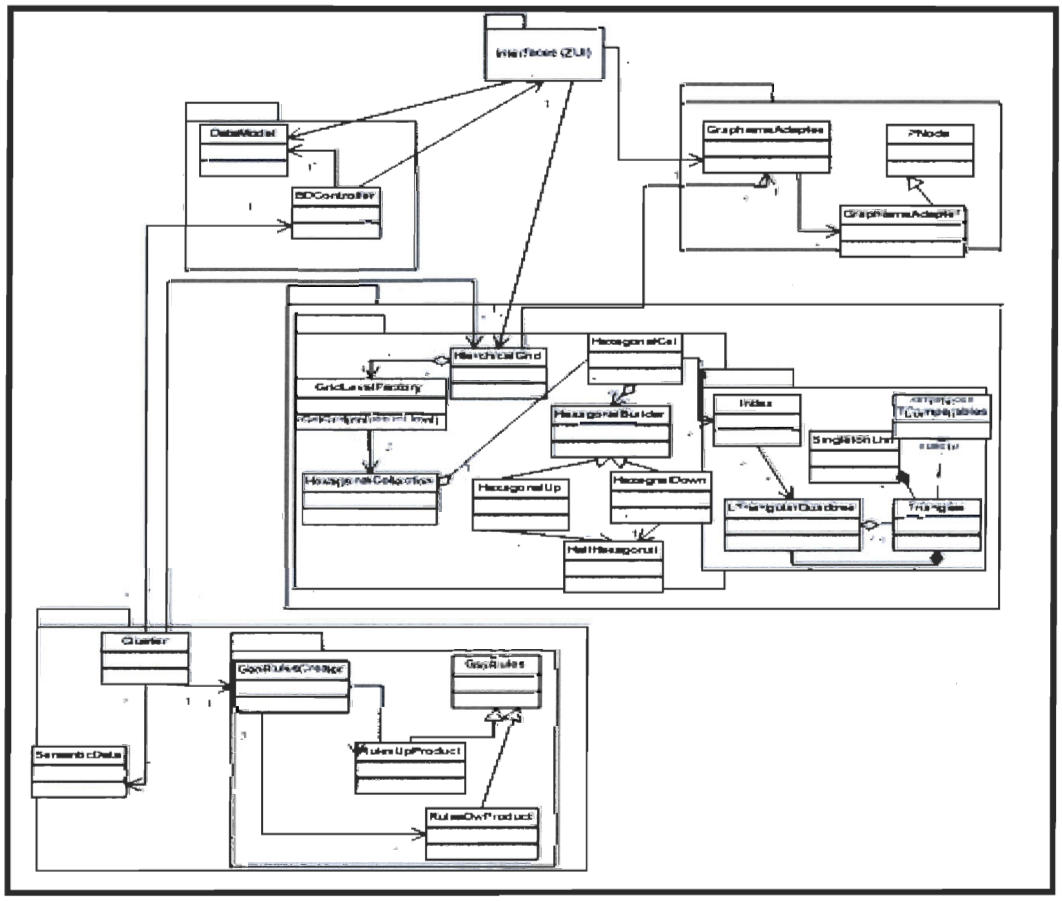


Figure 6.7 Modèle de classe du système

6.2.3 Conclusion

Dans cette section, nous avons présenté les détails techniques de notre prototype. Notre environnement technique est composé du langage C#, l'outil de développement, Visual

studio 2005, le logiciels MapInfo et le système de gestion de bases de données Access. Nous avons également présenté l'architecture entière de notre système. Finalement, nous avons présenté plus de détails au sujet de cette architecture, plus particulièrement les classes et les packages qui la composent en utilisant le formalisme UML. Dans ce qui suit, nous allons présenter notre algorithme de regroupement et les résultats que nous avons obtenus.

6.3 Démarche de regroupement des cellules

La création des clusters est importante afin de déterminer des zones possédant des propriétés sémantiques communes. Par exemple, dans notre cas, des clusters qui ont un déficit de voitures. De plus, à cause des particularités spatiales de notre grille, et plus particulièrement des cellules triangulaires, ces clusters ont également des propriétés spatiales, par conséquent, la question qui se pose est la suivante : Comment utiliser ces deux différentes propriétés afin d'avoir un regroupement qui se base à la fois, sur des critères spatiaux et sémantiques. Nous présentons deux approches. L'une est analogue aux techniques de clustering du Data Mining spatial. Quant à l'autre, elle est similaire aux automates cellulaires (voir chapitre2).

6.3.1 Approche par clustering

Dans cette section, nous allons présenter notre démarche pour faire du clustering. Premièrement, nous présentons comment nous avons spécifié un cluster dans notre cas. Ensuite, nous présentons les concepts de distances sémantique et géométrique entre deux clusters, et finalement nous présentons l'algorithme de regroupement.

6.3.1.1 Spécification d'un cluster

Un cluster est un regroupement de plusieurs cellules. Il possède des propriétés sémantiques et géométriques. Dans ce qui suit, nous présentons ses caractéristiques.

Un cluster possède des spécifications sémantiques, car chaque cellule contient :

- Les valeurs qui représentent la dimension sémantique d'intérêt. Par exemple, un potentiel de déficit de voitures.
- Un identifiant unique.

Chaque cluster possède également des propriétés géométriques à cause de :

- La forme triangulaire de ses cellules.
- L'ensemble de tous les sous-triangles d'un cluster forme un polygone.
- Chaque cluster est un polygone qui a un centroïde de coordonnées x et y.

Pour conclure, on peut considérer chaque cluster comme un polygone ayant un centroïde auquel on associe des valeurs sémantiques et c'est ce dernier qui représentera tout ce cluster.

6.3.1.2 Distance

La mesure des distances est importante. En effet, elle permet de décider si deux clusters sont sémantiquement ou géométriquement proches.

a- Distance Sémantique :

Nous appelons distance sémantique, la différence entre deux valeurs sémantiques de deux cellules ou de deux clusters (voir chapitre 2). Dans notre cas, on considère la différence des deux potentiels de chaque cellule ou de chaque cluster.

Soit P le potentiel d'un cluster c. La distance sémantique (Ester et al 1998) entre deux clusters ci et cj est :

$$\sum_{j=1}^{k}\sum_{i=1}^{n}\left\| P_{c_i} - P_{c_j} \right\|^2$$

Ces valeurs sont issues du jeu de données que nous avons présenté dans le chapitre 5 et que nous avons intégré à notre environnement. Ceci est similaire à la méthode de carroyage dans les *SIG*. En effet, les index (voir chapitre 4) des cellules triangulaires jouent le rôle de références spatiales qui référencent les données que nous avons préparées (voir chapitre 5). Les opérations de calcul de distances sémantiques vont s'appliquer sur ces données pour faire du regroupement et plus particulièrement, pour calculer la distance sémantique entre deux cellules/clusters. Rappelons que cette approche est similaire aux travaux de Paris (Paris, 2007). En effet, après avoir découpé son environnement, il l'informe en introduisant des données sémantiques par l'intermédiaire des nœuds conceptuels pour conserver de l'information afin d'exploiter l'abstraction spatiale de l'environnement dans un cadre de simulation. Quant à Samet, (Samet,1990), contrairement à Paris, elle utilise une approche seulement géométrique avec l'utilisation des opérations géométriques.

b- Distance géométrique:

Ce n'est autre que la distance euclidienne entre deux clusters. Du fait qu'on associe une sémantique aux centroïdes de ces polygones, ils deviennent alors des points qui représentent tous ces clusters. Ainsi, pour calculer la distance entre deux polygones, il suffit de calculer la distance entre leurs centroïdes respectifs. Cette distance est donnée de la façon suivante.

Soit A(*xa,ya*) et B(*xb,yb*) deux centroïdes de deux clusters *C1* et *C2*. La distance euclidienne *d* entre *A* et *B* est :

$$d(A,B) = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$$

Par ailleurs, en considérant un cluster comme un polygone et puisqu'il se décompose en des sous-triangles équilatéraux, il est facile dans ce cas de calculer son centroïde. En effet, les coordonnées du centroïde du polygone sont égales à la moyenne des coordonnées des centroïdes de ses sous-triangles équilatéraux.(Rappelons que les coordonnées du centroïde d'un triangle équilatéral est égal au deux tiers de la médiane de ce triangle).

Soit un triangle équilatéral *ABC,* le point *G(x,y)* son centroïde, *I* le milieu du segment [BC] et *AI* une médiane de *ABC*. Nous avons alors :

$$\overrightarrow{AG} = \frac{2}{3}\overrightarrow{AI}$$

Soit un polygone P, *C(x, y)* le centroïde de ce polygone et n le nombre des centroïde des cellules triangulaires de ce polygones *Gi (xi,yi)* de ses sous-triangles, la formule (Ester et al 1998) qui calcule les coordonnées de *C* est la suivante.

$$C(x,y) = \begin{cases} x = \sum_{i=1}^{n} \left(\frac{x_i}{n}\right) \\ y = \sum_{i=1}^{n} \left(\frac{y_i}{n}\right) \end{cases}$$

Après avoir présenté la définition d'un cluster et les distances que nous allons utiliser, nous allons présenter le mécanisme de l'algorithme de clustering.

6.3.1.2 Algorithmes de regroupement selon une approche de Data Mining

Nous nous sommes basés sur un algorithme de clustering par fusion. Cet algorithme se fait en deux étapes : la première est une étape d'initialisation pour trouver les premiers

noyaux. La deuxième est la phase de regroupement. Dans cette section, nous allons présenter chaque étape.

6.3.1.2.1 Initialisation

Nous allons associer des données sémantiques aux cellules à partir des index qui leur sont associés et par le fait même localiser les cellules qui correspondent à un ou plusieurs critères sémantiques sur la carte. Cette initialisation dans les algorithmes qui sont présentés dans le chapitre 2, est aléatoire. Dans notre cas, plutôt que de partir avec choix arbitraires, c'est-à-dire des index de cellules triangulaires aléatoires, nous avons considéré les deux heuristiques suivantes.

a) Cellules sémantiques :

Ce sont les cellules qui contiennent une valeur sémantique. En effet, au lieu de considérer chaque cellule de toute la grille comme un noyau initial à regrouper, nous considérons seulement les cellules dont les index sont égaux à ceux des critères sémantiques choisis et qui possèdent une valeur sémantique. Il s'agit également de localiser les cellules triangulaires d'intérêt sur la carte. Ceci restreint l'espace de recherche.

b) Les premiers clusters appartiennent au même hexagone :

Nous avons considéré l'adjacence directe entre les cellules triangulaires au sein du même hexagone. En effet, nous considérons seulement les cellules ayant une valeur sémantique, et qui appartiennent à un même hexagone. Ces cellules constitueront alors les premiers clusters.

Ces deux heuristiques offrent un avantage d'un point de vue performances et optimisent l'algorithme de regroupement. En effet, elles permettent de restreindre l'espace de recherche puisqu'au lieu de s'intéresser à toutes les cellules de la grille, on s'intéresse seulement à celles qui ont une valeur sémantique. De plus, considérer les cellules triangulaires d'un même hexagone comme premiers clusters nous évite de faire des itérations supplémentaires pour les regrouper. Nous avons un accès direct aux premiers clusters lors de la première itération puisqu'ils appartiennent au même hexagone. En conclusion, les premiers clusters sont ceux qui sont localisées sur la carte, qui appartiennent au même hexagone. Finalement, ces premiers germes sont sauvegardés dans une liste qui

sera utilisée dans la deuxième étape. L'algorithme de l'étape d'initialisation et ses fonctions sont représentés par les figures suivantes.

```
Initialiser (List List_Hex )
Entier cpt = 0                                          // compteur
Entier Id = 0                                           // identifiant du groupe
List ListItem                                           // List des item d'un cluster
Réel Moy                                                // valeur Moyenne des triangles
Pour chaque cellule hexagonale H dans List_Hex
Pour chaque Triangle T de H
        Si (T.ValSemantique = = vrai) Alors
            cpt++ et Moy += (T.VAL )                    // calculer la moyenne et nombre Item
            Ajouter T à ListItem
          FinSi
    FinPour
Moy = (Moy / cpt)
 C ←new cluster (ListItem, Moy)                         // Créer un nouveau cluster
 Point g =  C. GravityPolyGon ()
  C ← Id++
   Ajouter C à la liste des Clusters Clust_List
   Dessiner C
  Cpt =0
 FinPour
```

Figure 6.8 Algorithme d'initialisation

```
Point  GravityPolyGon (List ListTri)
Réel a = 0; b = 0,Point g
Entier nbTri ← obtenir le nombre des éléments de la List
Pour m =  0;  m  <  nbTri;  m++
g     =     GravityTriangle     ((Triangle) ListTri[m])
                a = a + g.X;   b = b + g.Y
FinPour
Si (nbTri != 0)
Retourner Point(a/nbTri), (b/nbTri)
Sinon  retourner null
```

Figure 6.9Algorithme de calcul du centroide d'un polygone

```
Point GravityTriangle (Triangle T)
Debut
        Point [] tab ← obtenir les points de T
        Point G = (Point) tab [0]
        Point D = (Point) tab [1]
        Point H = (Point) tab [2]
        Point I = Milieu (G, D)
        Point g = (2/3. ((Hx+Ix)), (2/3. (Hy+Iy))
        Retourner g
Fin
```

Figure 6.10 Algorithme de calcul du centroide d'un triangle

```
Point Milieu (Point A, Point B)
Début
Retourner  Point (((A.X + B.X)/2),((A.Y + B.Y)/2)
Fin
```

Figure 6.11 Algorithme de calcul du milieu d'un segment

Cet algorithme admet comme paramètres une seule liste qui représente les cellules hexagonales d'intérêt. En fait, ces cellules sont obtenues par une simple requête *SQL* entre nos tables d'adressage que nous avons présentées dans le chapitre 4, et les tables contenant les index des cellules triangulaires d'une dimension sémantique donnée. En effet, du fait que nos tables d'adressage associent un index d'hexagone dans la grille hexagonale aux index de ses 6 triangles, il est facile de retracer l'index de l'hexagone parent à partir des index des triangles enfants par une simple requête de jointure entre les deux tables. Un exemple d'une telle requête dans le cas ou la dimension sémantique choisie est le nombre de déficit de voitures par cellule est le suivant :

```
SELECT IdHex, RESIDGRHEX, DEFICITCEL FROM AdresseInterne, CDeficit
WHERE CDeficit.RESIDGRHEX = AdresseInterne.IdTri
```

Le résultat de cette requête est directement enregistré dans la structure de données linéaire *List_Hex* qui sera utilisée comme paramètres d'entrée de cet algorithme. Le but de ce dernier est de créer les premiers groupes en se basant sur les heuristiques que nous avons présentées plus haut. Pour ce faire, il parcourt la liste *List_Hex,* il extrait chaque hexagone de cette liste et vérifie si ses triangles contiennent une valeur sémantique. Si oui, il calcule la moyenne des cellules triangulaires qui formeront un premier cluster et il les ajoute à la

liste *Liste_Tri* qui servira d'objet interne au cluster. Également, nous associons à ce cluster la moyenne calculée, un identifiant unique de ce groupe fraîchement créé conformément aux spécifications que nous avons présentées au début de la section regroupement. La caractéristique la plus importante du cluster est son centroïde. Ce dernier est calculé à l'aide de la fonction *GravityPolygon.* Cette fonction fait appel aux routines qui sont présentées par les figures 6.9, 6.10 et 6.11. En effet, cette fonction traduit l'équation mathématique que nous avons utilisée pour calculer la distance géométrique. Elle admet comme paramètres une liste qui contient des triangles : ce sont les sous-triangles d'un polygone donné. Dans notre cas, cette fonction recevra *Liste_Tri* comme paramètre d'entrée. Par l'intermédiaire d'une boucle répétitive, cette fonction détermine le centroïde G de chaque triangle de *Liste_Tri* grâce à la fonction *GravityTriangle*, ensuite elle trouve la moyenne des centroïdes des triangles de *Liste_Tri* et ce, conformément à la formule que nous avons présentée dans la section précédente. Finalement, cette fonction retourne les coordonnées du centre de gravité de tout le polygone qui n'est autre que la somme totale des coordonnées x et y des centroïdes de chaque triangle, divisée par le nombre total *cpt* des triangles de la liste *Liste_Tri.* Quant à de la fonction *GravityTriangle,* elle extrait les trois sommets d'un triangle T passé comme paramètre et calcule le milieu I du segment qui représente la base de ce triangle. Ensuite, elle calcule les *2/3* du segment issu du sommet d'en haut qui est perpendiculaire à la base de ce triangle, et qui le coupe en I. Le point trouvé est le centre de gravité du triangle T. Après cette étape d'initialisation, l'étape de regroupement est entamée. Nous présentons cette étape dans la section suivante.

6.3.1.2.2 Algorithme de regroupement

Dans cette partie, nous présentons la deuxième étape de notre approche qui assure le regroupement des différentes cellules. Nous allons d'abord expliquer le déroulement de cette étape avant de présenter les algorithmes et aborder plus en détail tout le processus de regroupement. Comme les premiers noyaux qui ont été regroupés lors de l'étape d'initialisation, ils sont réutilisés dans cette étape pour qu'à leur tour, ils puissent être clustérisés de nouveau. Pour ce faire, nous devons fixer les contraintes liées aux aspects sémantiques et géométriques. En effet, nous devons vérifier tout d'abord la proximité sémantique pour voir si les deux clusters possèdent les mêmes propriétés sémantiques. Si cette condition est vérifiée, nous devons examiner la proximité géométrique pour voir à

quel point les clusters sont proches l'un de l'autre. Pour ce faire nous calculons le centroïde de chaque cluster. Ensuite nous calculons la distance euclidienne entre ces deux centroïdes. Si cette distance est inférieure à la distance autorisée par l'usager au lancement de l'application, nous procédons à un regroupement. Dans ce cas, nous devons déterminer, les coordonnées du nouveau centroïde du cluster résultant. Nous devons également trouver la nouvelle moyenne des valeurs sémantiques du cluster résultant et l'associer à son nouveau centroïde. Finalement, les listes des cellules triangulaires appartenant à ce cluster sont mises à jour. C'est ainsi, que nous ferrons évoluer notre système et chaque entité évoluera au cours des itérations de l'algorithme. L'algorithme principal *Cluster* ainsi que ses fonctions sont représentés respectivement dans les figures suivantes.

```
Cluster(Color couleur1, Color couleur2, Réel d, entier nb)
List grille ← obtenir la grille
 List listId ← nouvelle liste
Pour  l = 0; l < nb; l++
 Pour  ( j = 0; j < Grp.Count; j++)
   Cluster C1 = (Cluster)Grp [ j ];        j = j + 1                    // obtenir un cluster
    Pour ( h = j; h < Grp.Count; h++)
      Cluster C2 = (Cluster) Grp [ h ];                                 // obtenir un autre cluster
          Si (C1.MOY >= C2.MOY))                                        // condition sémantique
            Si ((Distance (C1.centre, C2.centre)) < d)                  // condition géométrique
                    Point M = MilieuCentroide (C1, C2);
                    List T = FindCel (G, M, G1, C2, grille, d, listId);
                    Dessiner les cellules absorbées.
                     Dessiner le cluster C1 avec couleur1
                    Dessiner le cluster C2 avec couleur 2
                    Copy (C1, C2)                                       // copier C2 dans C1
                    Enlever l'élément C2 à la position h de Grp         // mise à jour de la
liste
                FinSi
              FinSi
            FinPour
          FinPour
        FinPour
```

Figure 6.12 **Algorithme de regroupement**

```
Copy (ClusterC1, Cluster C2)
  Chaine Nom = C1.Nom + C2.Nom                    // concaténer les deux noms
  Réel Moy = (C1.Moy + C2.Moy)/2                  // calculer la nouvelle moyenne
 Point centre = Milieu (C1.centre, C2.centre)     // calculer le nouveau centroide (voir figure
6.8d)
 tailleT ← Obtenir le nombre de triangles de la liste C2.List
  Pour (i = 0; i < tailleT; i++)                  // copie les triangles  dans une liste
C2.List[i] à C1.List
    Entier tailleH ←Obtenir le nombre d'hexagones de la liste C2.LidHex
  Pour (j = 0; j < tailleH; j++)                  // copier les hexagones dans une liste
   Ajouter C2.ListH[i] à C1.ListH
 Fin
```

Figure 6.13 Algorithme pour ajouter un cluster à un autre

```
Réel Distance (Point A, Point B)
        retourner Math.Sqrt(((B.X - A.X) * (B.X - A.X)) + ((B.Y - A.Y) * (B.Y - A.Y)))
Fin
```

Figure 6.14 Algorithme pour calculer la distance entre deux points

```
List FindCel(Graphisme G, Point Centroide, Cluster Gr1 et  Gr2, List grille, distance)
Début
List listH1 ← Obtenir les index des hexagones de Gr1
List listH2 ← Obtenir les index des hexagones de Gr2
ListTriangle = Listdistance = ListTri← Nouvelle liste   // création de listes
Réel dist = 0
Entier mini  ← Mini de listH1              // trouver l'index minimal
Entier max ← Maxi de listH2                //trouver l'index maximal
// Mini et maxi sont des routines de C#
List range = grille.GetRange(mini, (max - mini) + 1)  // obtenir les éléments entre min et max
Pour (j = 0; j <  taille de range ; j++)
HexagonalCel h ← range[ j ]
List triList ← obtenir la liste des triangles de h
Point G ← calculer le centre de gravité du polygone tri_List     // voir figure 6.8b
Booléen exp11 = ((G.Y <= Gr1.centre.Y) & (G.Y >= Gr2.centre.Y))
Booléen exp22 = ((G.Y >= Gr1.centre.Y) & (G.Y <= Gr2.centre.Y))
Booléen exp33 = ((Gr1.centre.Y = = Gr1.centre.Y))
Si (exp11 || exp22)
List listTri ←obtenir la liste des triangles de h
Pour (int i = 0; i < listTri.Count; i++)
Triangle T = listTri[i]
Point g ← calculer le centre de gravité de T                  // voir figure 6.8c
dist ← calculer la distance entre les points Centroide et g // voir figure 6.9c
Si (dist <= distance)  ajouter T à ListTri                 // proximité géométrique
Fin Pour
Fin Si
Fin Pour
Retourner ListTri
Fin
```

Figure 6.15 Algorithme de recherche des cellules entre deux clusters

L'algorithme principal présenté par la figure 6.12 est celui qui est responsable du regroupement et il fait appel aux autres fonctions (voir figures 6.13, 6.14, 6.15). Il admet comme paramètres deux couleurs pour colorier les cellules ayant une forte et une faible densité, le nombre *nb* qui représente le nombre d'itérations pour le regroupement, et la distance *d* qui est choisie par l'utilisateur. Cette dernière permettra à l'usager de définir la portée du regroupement, c'est-à-dire la distance minimale jusqu'à laquelle des cellules peuvent se regrouper.

L'algorithme parcourt la liste des premiers noyaux qui sont créés lors de l'étape d'initialisation. Ensuite, il extrait un cluster *C1*, et il parcourt la même liste pour extraire un

autre cluster *C2*. À chaque fois, il vérifie la condition sémantique entre *C1* et *C2*, c'est-à-dire qu'il vérifie si le potentiel ou la valeur sémantique de *C1* est supérieur à celui de *C2*.
En fait, par valeur sémantique on sous-entend le chiffre qui correspond à un critère sémantique et qui est associé à la cellule ou au cluster.
Dans le cas ou cette condition est satisfaite, il vérifie alors la condition géométrique, c'est-à-dire, il vérifie la proximité géométrique entre les deux clusters par rapport à la distance *d* qui est fixée par l'usager. Si cette condition est à son tour satisfaite, l'algorithme procède à un regroupement en ajoutant les cellules qui se situent entre *C1* et *C2* au cluster résultant qui a la plus grande valeur sémantique par l'entremise de la fonction *FindCel*(voir figure6.11). Il dessinera ensuite le cluster fraîchement créé et il fusionnera *C1* et *C2* dans un seul cluster émergeant grâce à la fonction *Copy* (voir figure6.10). Finalement, il enlève *C1*et *C2* de la liste. La fonction *Copy*, comme nous l'avons mentionné, elle copie deux clusters dans un autre cluster émergeant. Elle calcule la nouvelle moyenne des valeurs sémantiques de ce cluster, le nouveau centroïde. Elle fusionne aussi les listes internes qui contiennent les cellules triangulaires de chaque sous-cluster en une seule liste qui représentera l'ensemble des cellules de ce cluster. Cette dernière est également mise à jour au niveau de la fonction *FindCel*. En effet, cette fonction détermine le minimum et le maximum des index des cellules des deux clusters précédents *C1* et *C2*. Ainsi, elle définit les frontières de la zone dans laquelle il y aura des cellules à absorber. Ensuite, pour chaque cellule hexagonale appartenant à cette zone, on calcule son centre de gravité et on vérifie sa proximité géométrique par rapport à la même distance d qui est fixée par l'usager. La satisfaction de cette condition entraine l'ajout de ces cellules à la liste d'items du cluster émergeant. Notons également, qu'à chaque fois ou nous devons évaluer la proximité géométrique entre deux centroïdes et la distance *d*, nous avons recours à la fonction *Distance* (voir figure 6.11) qui applique la formule de mesure de distance euclidienne que nous avons présentée au début de cette section et retourne une valeur réelle qui est convertie dans la même échelle que *d*. Dans la section suivante, nous allons présenter nos tests et nos résultats obtenus pour valider notre démarche.

6.3.2 Résultats qualitatifs de l'approche par clustering

Lors d'un premier test, les cellules initiales d'intérêt sont localisées et leurs premiers centroïdes sont calculés. Elles sont colorées en orange. Ensuite, on vérifie la condition sémantique de *Cluster 1* par rapport à *Cluster2.* Comme *Cluster1* a une valeur sémantique élevée, on le colorie en bleu foncé, tandis que *Cluster2* est colorié en bleu clair. (Figure 6.16). Finalement, on fixe une distance géométrique égale à 15 mètres tout en vérifiant la proximité du centroïdes du sous-cluster (figure 6.17). Le cluster émergeant est celui qui a un contour noir.

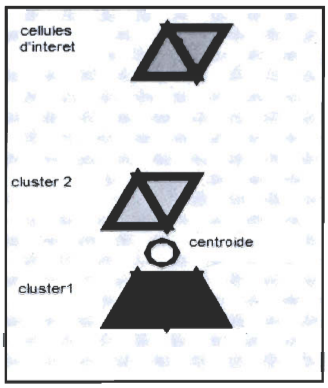


Figure 6.16 Déroulement de l'algorithme

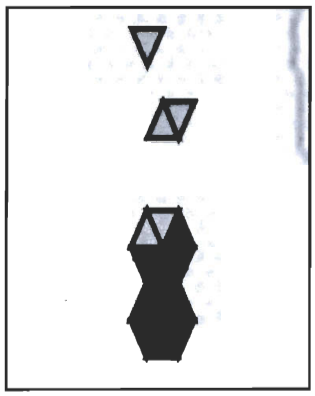


Figure 6.17 Regroupement

Les figures suivantes présentent une vue d'ensemble et une vue élargie du résultat obtenu. Nous remarquons des clusters qui forment des zones. Ces clusters contiennent des couleurs bleu foncé et pâle. Nous avons utilisé la légende suivante pour illustres les résultats.

A : En noir, deux clusters formés par l'algorithme présenté par la figure 6.12.

B : Deux cellules localisées sur la carte par l'algorithme d'initialisation (voir figure 6.8)

C : Un texte qui identifie le cluster. Il joue le rôle d'un identifiant unique de ce cluster.

D : Cellules avec une valeur sémantique faible.

E : Cellules avec une valeur sémantique élevée.

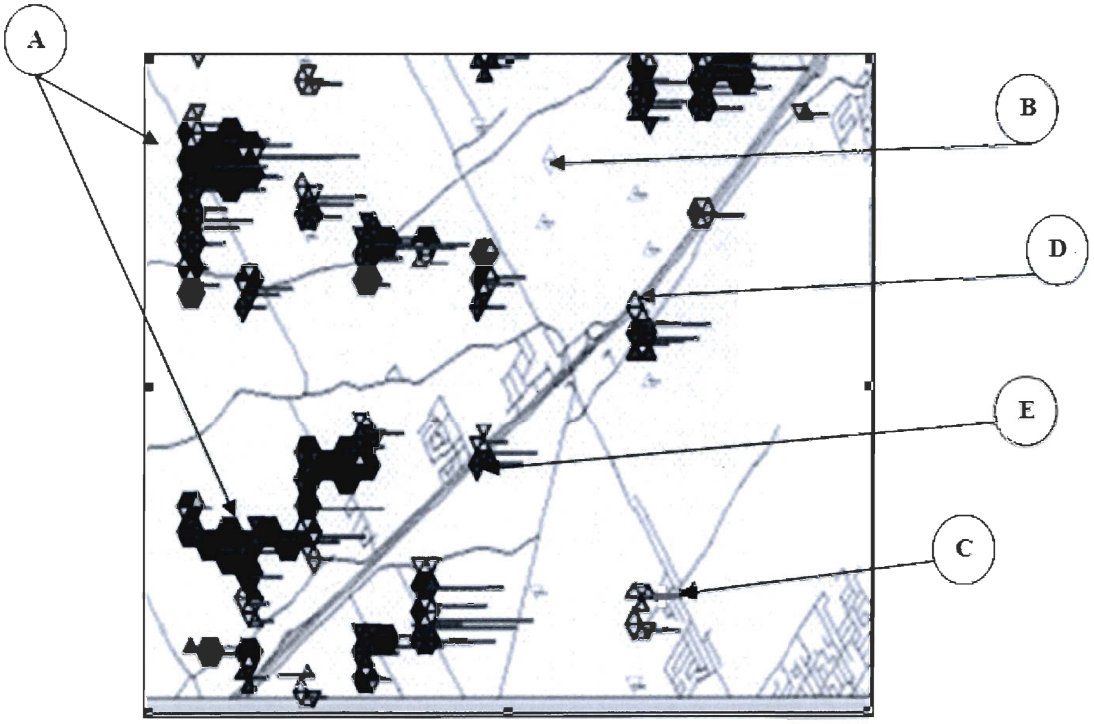


Figure 6.18 Vue élargie du résultat du clustering

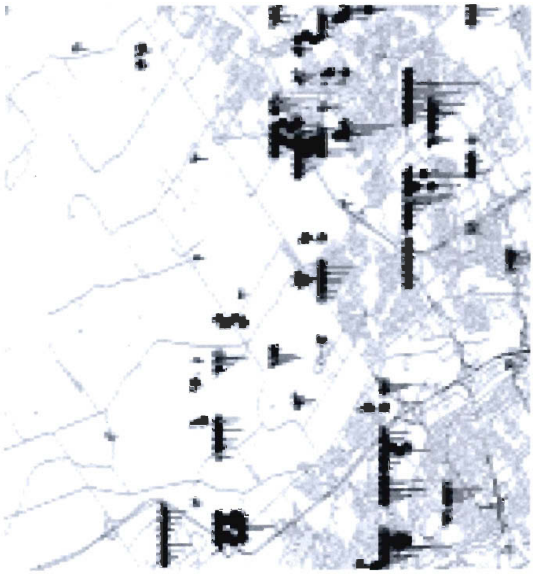


Figure 6.19 Vue d'ensemble des clusters du nombre d'adolescents en 2002

Dans un deuxième test, nous nous sommes intéressés à la dimension sémantique *déficit de véhicules* puisqu'elle est au cœur même de notre problématique. Pour cette même dimension nous allons changer les paramètres d'entrée. Nous changeons les couleurs pour montrer justement l'intérêt de les avoir comme paramètres d'entrée. En effet, nous avons choisi la couleur rouge pour les cellules qui ont des valeurs sémantiques élevées et la couleur jaune pâle pour celles qui ont des valeurs sémantiques faibles. Les identifiants des clusters sont en vert. La couleur noire, ainsi que le contour noir sont également utilisées pour désigner un cluster émergent. Nous remarquons des clusters de couleur noire qui sont formés sur la carte de la ville de Québec (voir figures suivantes). Chaque cluster possède des densités en termes de déficit de voitures. Dans un même cluster, deux couleurs peuvent cohabiter, c'est-à-dire, il présente les cellules ayant une forte et une faible valeur sémantique. Dans cet exemple, ce sont les couleurs rouges et jaune pâle. Rappelons également, que l'annexe C présente un guide-utilisateur qui montre l'utilisation de notre outil pour paramétrer le processus de clustering. Ensuite, nous gardons les mêmes paramètres d'entrée précédents et nous changeons seulement la distance géométrique. Ceci nous permettra de trouver la distance-seuil à partir de laquelle, les clusters émergents seront trop larges et par conséquent non significatifs. La figure 6.20 montre les résultats d'une distance de 10 mètres.

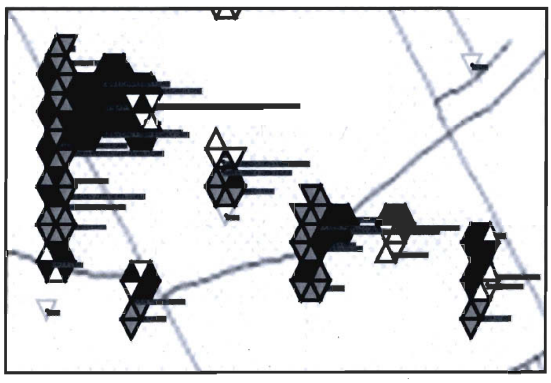


Figure 6.20 Résultat du clustering d'une distance de 10 mètres

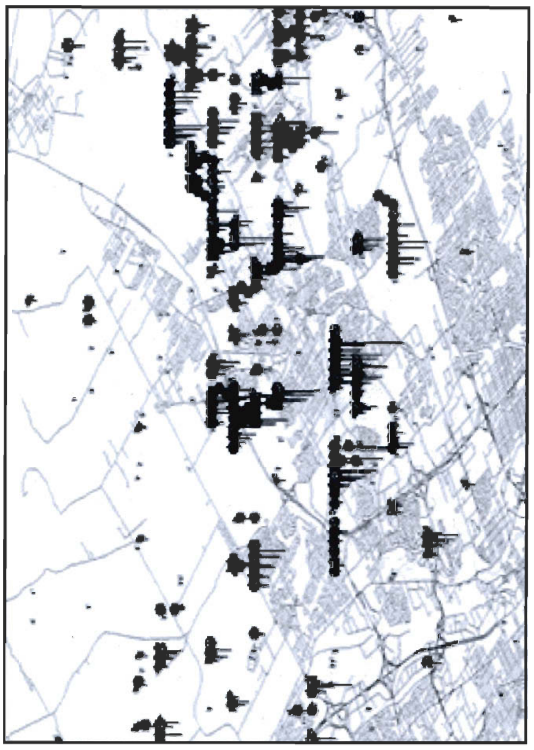


Figure 6.21 Vue élargie du résultat de clustering de 10 mètres

Nous avons considéré la distance de 10 mètres comme distance de base et nous allons l'augmenter de 5 mètres à chaque test pour essayer de déterminer empiriquement (par simple observation) à partir de quelle distance nous allons obtenir des clusters trop larges. Ensuite nous allons diminuer cette distance de base de 5 unités pour voir à quel point les clusters sont significatifs. Tout d'abord, on considère une distance de 15 mètres (figures suivantes).

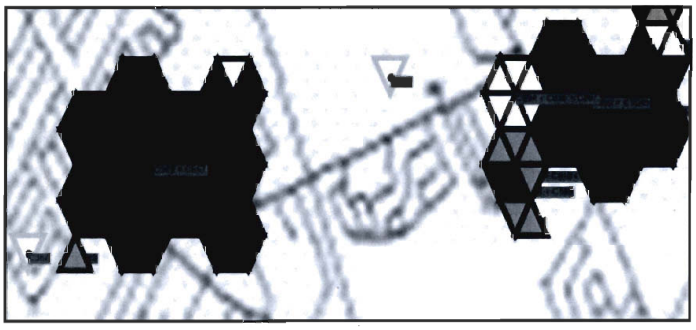


Figure 6.22 élargie du résultat du clustering d'une distance de 15 mètres

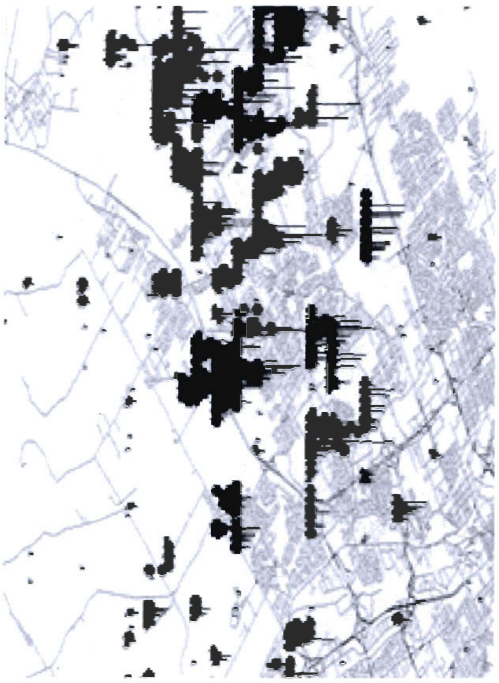


Figure 6.23 Vue d'ensemble du résultat du clustering d'une distance de 15 mètres

Ensuite, nous considérons une distance de 20 mètres. Les résultats obtenus (figure 6.24 et 6.25) montrent les résultats obtenus.

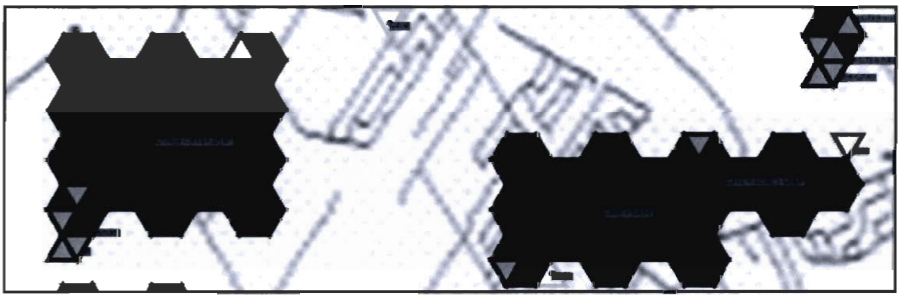


Figure 6.24 Vue élargie du résultat du clustering d'une distance de 20 mètres

Nous remarquons que les clusters sont de plus en plus larges entre 10 à 20 mètres. En effet, le nombre de cellules entre deux cellules de couleurs différentes augmente à chaque nouvelle distance plus grande que 10 mètres. Ceci résulte de l'algorithme puisqu'il va regrouper à chaque fois plus de cellules. Nous pouvons considérer que la distance 20 mètres est la distance limite à partir du quelle le clustering n'est plus significatif.

Après avoir déterminé cette distance, nous allons essayer de déterminer la distance qui représente le seuil minimal pour le clustering. Premièrement, on retranche 5 mètres de la distance de base (10 mètres) pour avoir une distance de 5 mètres. Les figures 6.14a et 6.14b montrent les résultats trouvés. Nous remarquons que les clusters sont moins larges que ceux des figures précédentes. Les cellules colorées sont plus proches les unes des autres, par contre dans les figures 6.22 et 6.24 elles sont plus éloignées.

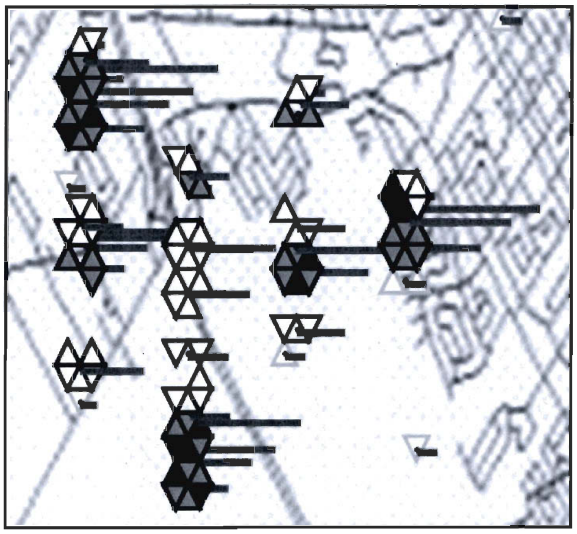


Figure 6.25 Vue élargie du résultat du clustering d'une distance de 5 mètres

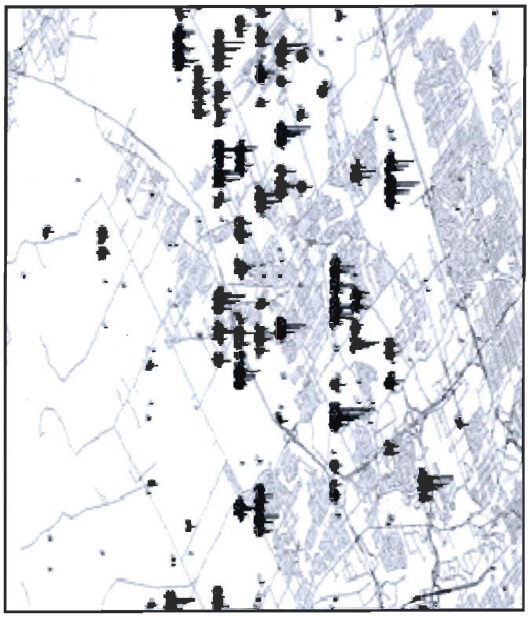


Figure 6.26 Vue d'ensemble du résultat du clustering d'une distance de 5 mètres

Nous remarquons dans la figure 6.25 les cellules sont très proches les unes des autres. Il n'existe que quelques cellules de couleur noire entre les cellules d'intérêts. On peut déduire alors que la borne limite inférieure du clustering est une distance égale à 5 mètres. À la suite de ces tests, nous concluons qu'en dessous de 5 et au-delà de 20 mètres, les résultats du clustering ne sont pas significatifs. La section suivante présente une étude quantitative de ces résultats.

6.3.3 Résultats quantitatifs de l'approche par clustering

La section 6.3.3 a présenté les résultats de manière qualitative. Dans cette section, nous avons trouvé que les distances entre 5 et 20 mètres sont des seuils pour le clustering. Dans cette section, nous allons élaborer une étude quantitative des résultats obtenus. Tout d'abord nous présentons notre démarche. Ensuite nous présentons les différentes courbes obtenues.

6.3.3.1 Démarche

Le but est de déterminer le taux de regroupement des cellules en fonction des indicateurs suivants : nombre de cellules totales d'un cluster, nombre de cellules avec faible et forte valeurs sémantiques, le nombre des cellules libres (non regroupées), et le nombre de cellules non significatives absorbées (appelées 'cellules NSA' dans la suite) lors du processus de clustering. Pour ce faire, nous avons placé quatre variables compteurs au niveau de l'algorithme présenté par la figure 6.12 de façon à compter ces nombres lors de l'exécution du programme. Ensuite, nous avons encapsulé ces variables dans un objet *cluster* afin d'enregistrer la croissance des clusters en termes de nombres de cellules au cours du clustering. Ainsi, chaque cluster devient une entité qui enregistre son propre taux de croissance au cours du temps, c'est-à-dire le nombre de cellules significatives (avec valeur sémantique) et le nombre de cellules NSA. Finalement, pour chaque distance (5, 10, 15 et 20 mètres) nous avons exécuté le programme et nous avons enregistré les nombres des cellules qui reflètent le taux de croissance des clusters dans une base de données grâce à la fonction *INSERT* de *SQL*. Ces données représentent les données fines. Nous avons traité ces données de façon à obtenir des ratios (nombre de cellules significatives et NSA divisé

par le nombre total de cellules) et nous les avons organisés dans des documents *Excel* afin de créer des courbes statistiques. Notons que tout ce processus pourrait être intégré à l'application elle-même via les composantes logicielles *COM/COM+* que la plateforme *.NET*. On pourrait ainsi offrir une fonctionnalité d'analyse de la qualité du regroupement. Nous ne l'avons pas fait par manque de temps. Cela fait partie des prolongements de ce travail. Dans la section suivante, nous allons présenter et discuter les courbes obtenues.

6.3.3.2 Courbes statistiques et résultats

Les courbes de cette section représentent les résultats de l'étude quantitative que nous avons élaborée pour déterminer la qualité du clustering. Ces diagrammes sont des courbes, des histogrammes et des tableaux statistiques. Pour les courbes, elles représentent les ratios de cellules significatives par rapport au nombre total de cellules par cluster. Les histogrammes représentent la taille des clusters formés en termes de nombre total de cellules. Finalement, les tableaux contiennent les taux des cellules qui se sont regroupées et celles qui ne le sont pas et ce, après l'exécution de l'algorithme de regroupement (voir figure 6.12). Sur l'axe des x on trouve les ratios des cellules significatives par clusters. C'est la somme des cellules avec une forte valeur sémantique et celles avec une faible valeur sémantique. Sur l'axe des y nous avons le nombre total de cellules, c'est-à-dire la somme des cellules significatives et les cellules NSA. Les carrés représentent les ratios des cellules significatives par rapport au nombre total des cellules des clusters obtenus. Également, nous avons élaboré d'autres courbes qui déterminent une moyenne. Nous avons représenté cette moyenne par une ligne noire. Quant aux histogrammes, on trouve sur l'axe des y le nombre total des cellules d'un cluster. Ainsi, une barre de valeur 10 par exemple, représente un cluster contenant 10 cellules au total. Ceci est représentatif et nous permet de constater la taille des clusters obtenus. Nous avons considéré deux indicateurs pour mesurer la qualité des résultats du regroupement. Nous considérons la cohésion des clusters, c'est-à-dire à quel point ces clusters sont liés d'un point de vu sémantique. Cette cohésion est exprimée en fonction de la somme des cellules significatives divisé par le nombre total des cellules, ou encore les ratios. En effet, plus la proportion de ces cellules significatives est importante plus ces clusters sont cohésifs. Nous considérons également le nombre de cellules total par cellule qui indique la taille de ces clusters. L'objectif est de trouver la

distance qui maximise le nombre de clusters formés tout en garantissant une cohésion moyenne optimale pour ces clusters.

1) Distance de 5 mètres :

Les figures suivantes représentent les résultats quantitatifs pour un clustering de 5 mètres.

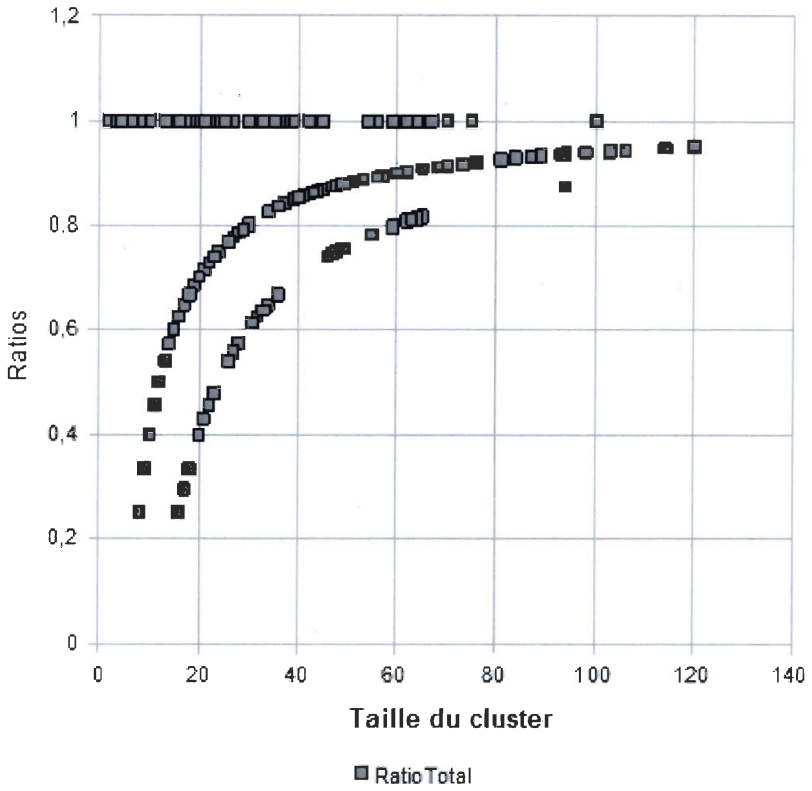


Figure 6.27 Courbe du regroupement d'une distance de 5 mètres

Indicateurs	Valeurs	Pourcentage
Moyenne des tailles des clusters	24	-
Cohésion moyenne des clusters	0,84	84%
Nb Total des cellules avant regroupement	2276	100%
Nb des cellules regroupées	786	34,36%
Nb Total de clusters créés après le regroupement	551	-
Nb Total des cellules libres après regroupement	1490	65,46%

Tableau 6-2 Tableau statistique du regroupement de 5 mètres

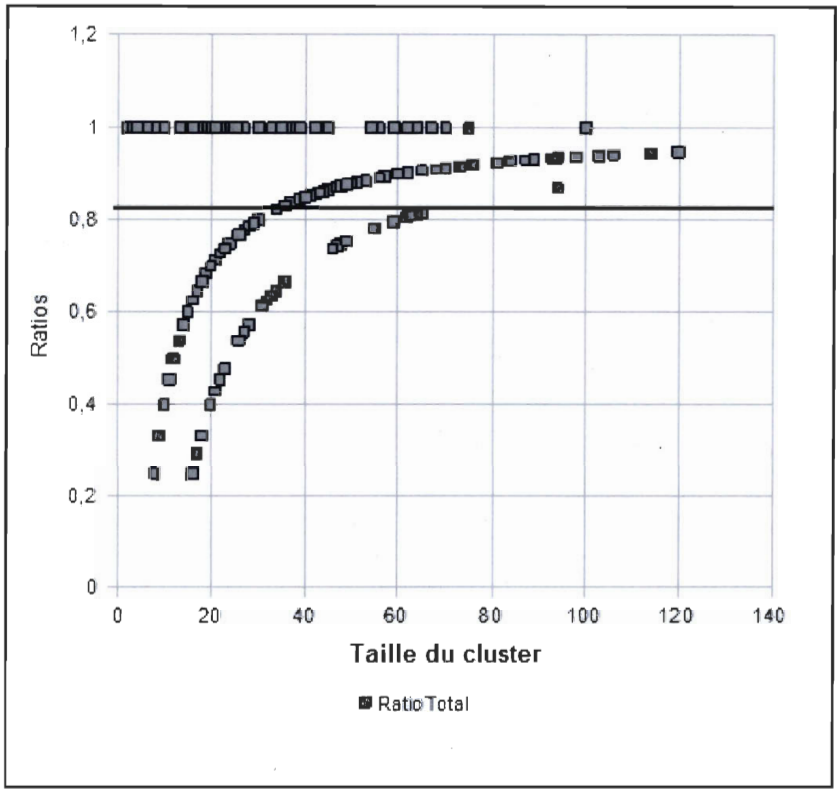


Figure 6.28 Courbe avec 84% de moyenne de cohésion pour 5 mètres

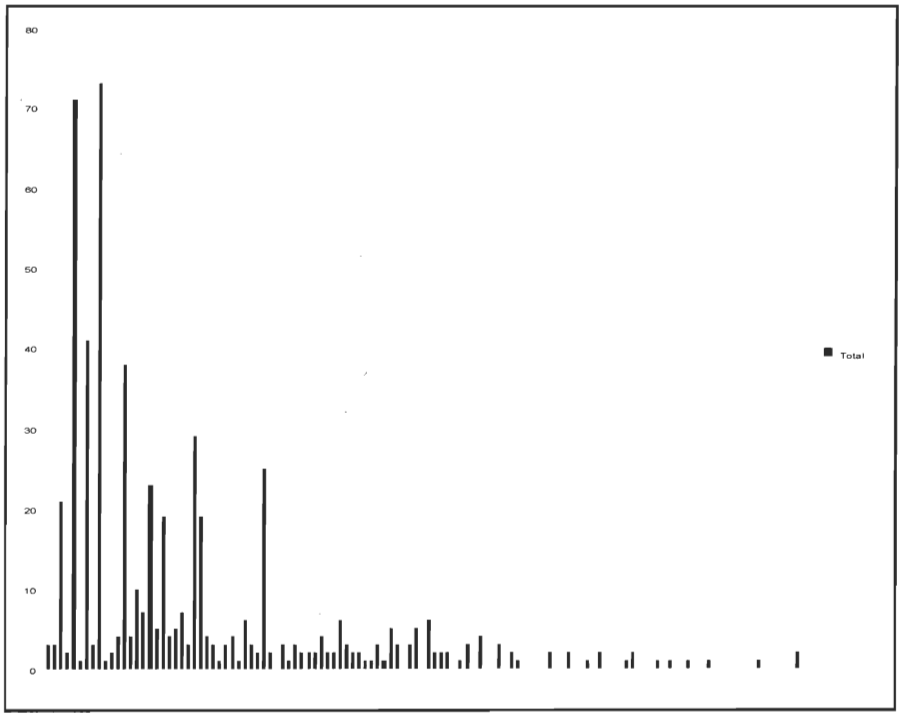


Figure 6.29 Histogramme du nombre de cellules total par cluster de 5 mètres

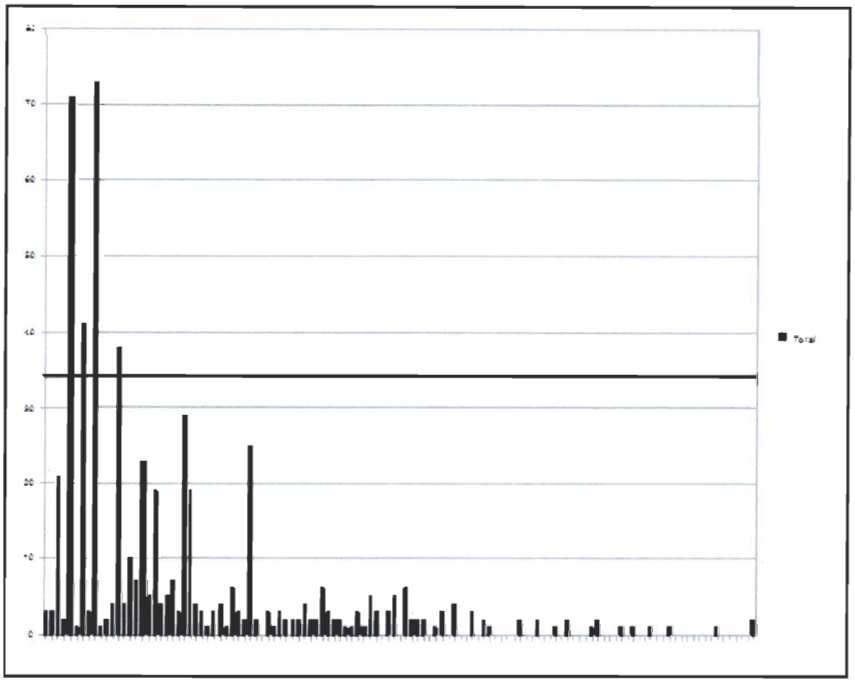


Figure 6.30 Moyenne de 24 cellules total par cluster de 5 mètres

Dans la figure 6.27, les carrés représentent les clusters. Nous remarquons qu'il y a un grand nombre de clusters avec des ratios de 100% de cellules significatives, c'est-à-dire le nombre de cellules absorbées et faible. Ceci est similaire aux résultats de la figure 6.28 où on trouve des clusters de 5 mètres ont un faible nombre de cellules noires. Cependant, il y a 65,46% de cellules libres qui ne se sont pas regroupées. Pour les autres cellules regroupées, les clusters ne sont pas larges. En effet, la figure 6.29 montre que la majorité des clusters ont une taille qui ne dépasse pas les 10 cellules au total. Quant au nombre total de clusters créés après le regroupement, il est égal à 551. La moyenne de la cohésion des clusters est égale à 84%. Elle est la plus grande parmi les autres distances. La moyenne de la taille de ces clusters est égale à 24. Nous pouvons conclure que pour une distance de 5 mètres, la cohésion est forte pour la plupart des clusters.

2) Distance 10 mètres :

Les figures suivantes représentent les résultats quantitatifs pour un clustering de 10 mètres.

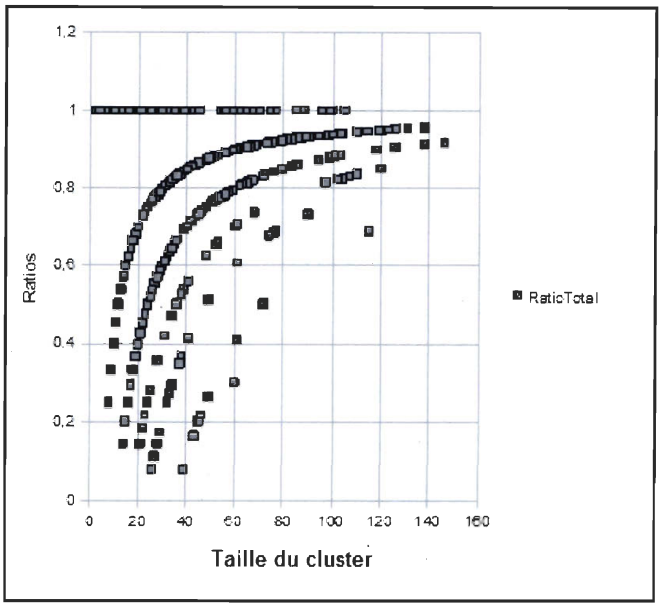


Figure 6.31 Courbe du regroupement d'une distance de 10 mètres

Indicateurs	Valeurs	Pourcentage
Moyenne des tailles des clusters	32	-
Cohésion moyenne	0,76	76%
Nb Total des cellules avant regroupement	2276	100%
Nb des cellules regroupées	1047	46%
Nb Total de clusters créés après le regroupement	789	-
Nb Total des cellules Libres après regroupement	1229	54%

Tableau 6-3 Tableau statistique du regroupement de 10 mètres

Pour une distance de 10 mètres, nous remarquons que le nombre des clusters a augmenté (Voire figure 6.31). Nous remarquons que par rapport aux clusters des figures précédentes, la majorité des nouveaux clusters ont des ratios situés entre 20 et 80% de cellules significatives pour un nombre total de cellules situé entre 20 et 140 cellules. Ceci indique que le nombre de cellules NSA est plus grand. Toutefois, il y a encore 54% de cellules significatives libres qui ne se sont pas regroupées. Cependant, il y a un gain en termes de cellules regroupées de 11% par rapport à celui de 5 mètres. Quant aux clusters formés, ils ne sont pas trop larges encore. En effet, la figure 6.33 montre que nous avons obtenu une moyenne de 32 cellules par cluster. La figure 6.34 montre quelques clusters dont le nombre

de cellule total dépasse les 60 cellules. Quant au nombre total de clusters créés après le regroupement, il est égal à 789. Il a augmenté par rapport à celui qui figure dans le tableau 6.2. L'indicateur de cohésion indique une moyenne de 76%. Elle est moindre que la précédente, mais elle reste significative. On peut conclure que pour 10 mètres, les clusters sont moins cohésifs que ceux de 5 mètres.

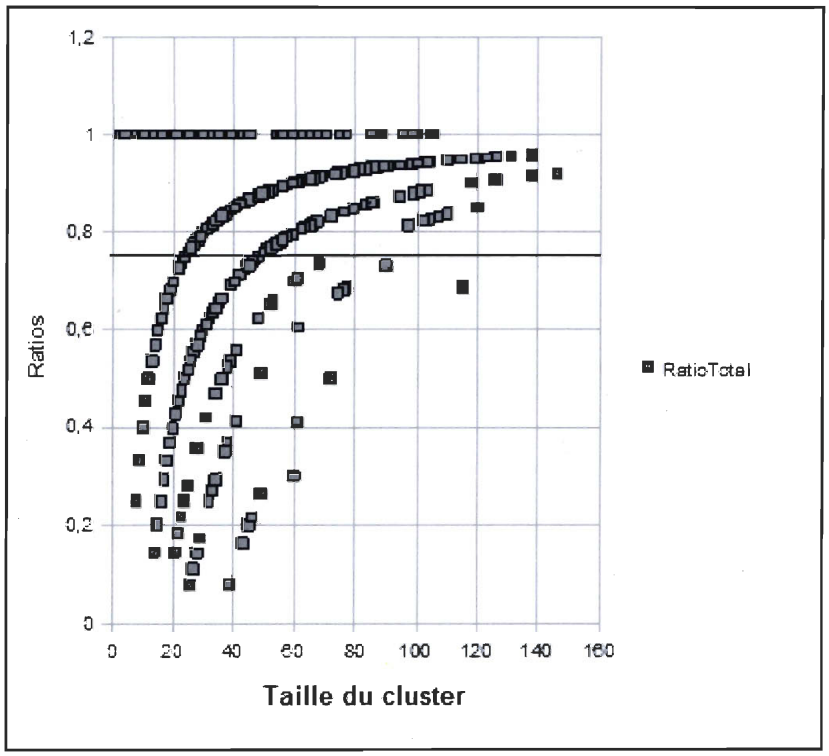


Figure 6.32 Courbe avec 76% de moyenne de cohésion pour 10 mètres

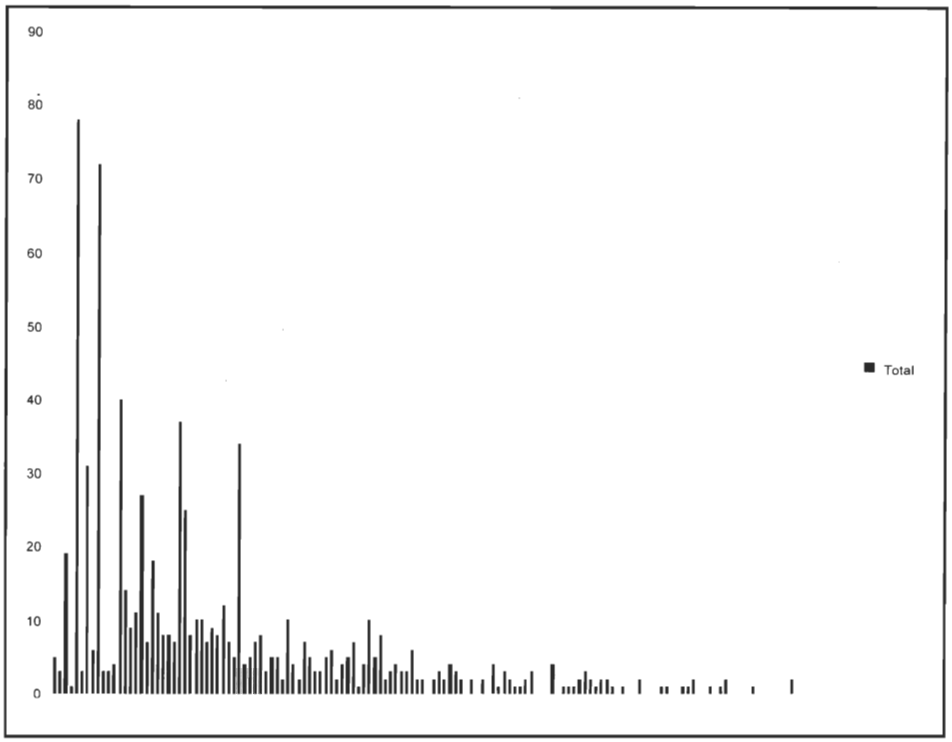


Figure 6.33 Histogramme du nombre de cellules total par cluster de 10 mètres

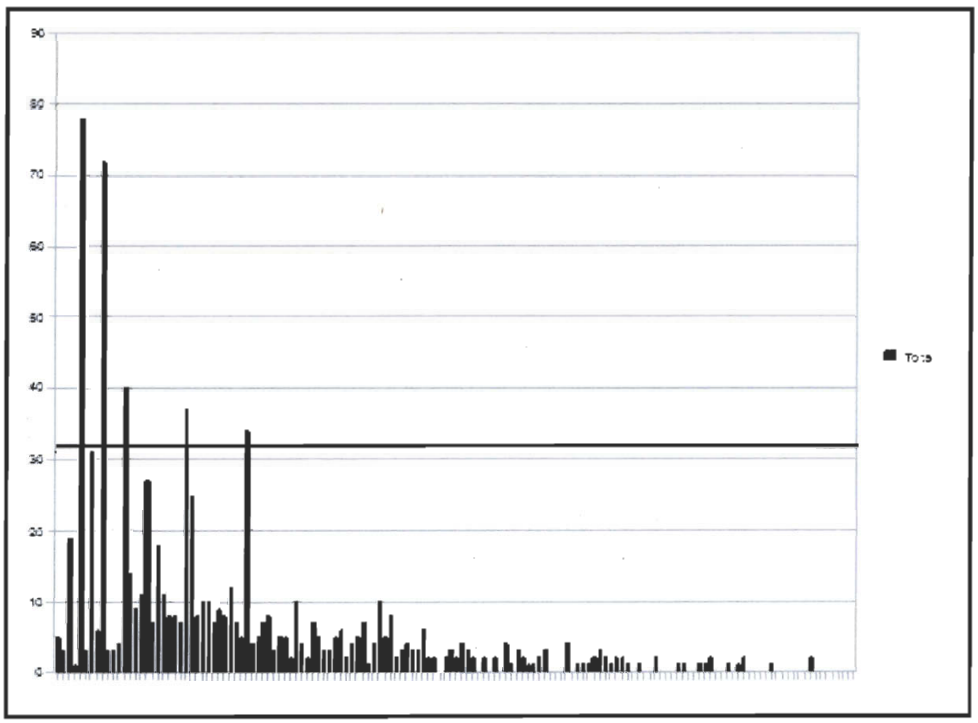


Figure 6.34 Moyenne de 32 cellules total par cluster de 10 mètres

3) Distance de 15 mètres

Pour une distance de 15 mètres, nous remarquons que le nombre des clusters a encore augmenté (voir figure 6.35). On retrouve 1191 cellules regroupées dans ce cas, par rapport à 1047 cellules pour une distance de 10 mètres, ce qui représente une augmentation de 7,3%. Nous remarquons que par rapport aux clusters de la figure 6.31, les nouveaux clusters ont pour la plupart des ratios situés entre 10 et 90% de cellules significatives pour un nombre total de cellules situé entre 20 et 160 cellules. Ceci indique que le nombre de cellules NSA a augmenté. En revanche, il y a encore 47,7% de cellules significatives libres. Le taux de regroupement d'une distance de 15 mètres et meilleur que celui des distances de 5 et de 10 mètres. Il y a un gain de 17% par rapport au clustering de 5 mètres et 6,3% par rapport au clustering de 10 mètres. Pour les cellules qui sont regroupées, la figure 6.37 montre que la majorité des clusters ont une taille inférieure à 15 cellules au total. Cependant, les clusters sont plus larges que ceux de la figure 6.33. Le nombre de clusters total qui sont créés après le regroupement il est égal à 926.

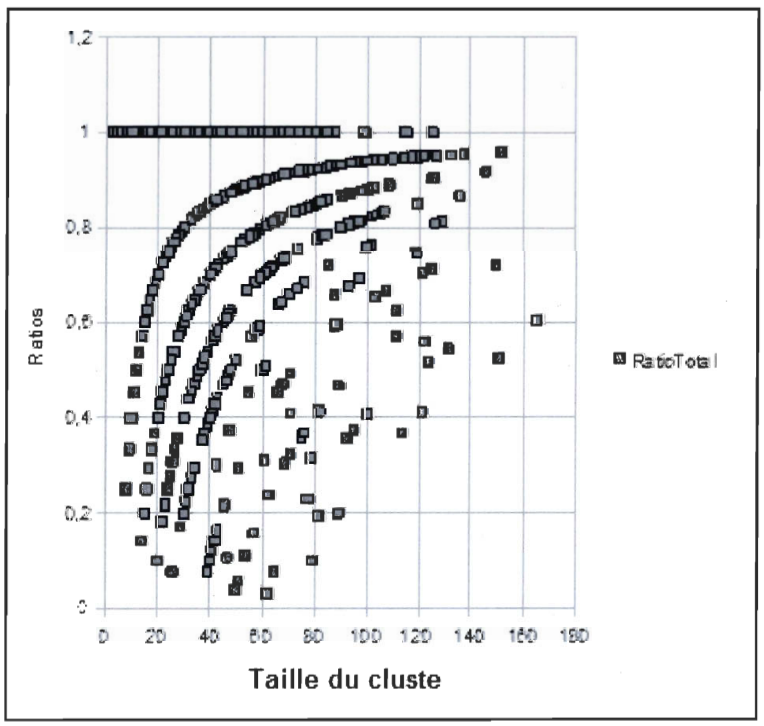


Figure 6.35 Courbe du regroupement d'une distance de 15 mètres

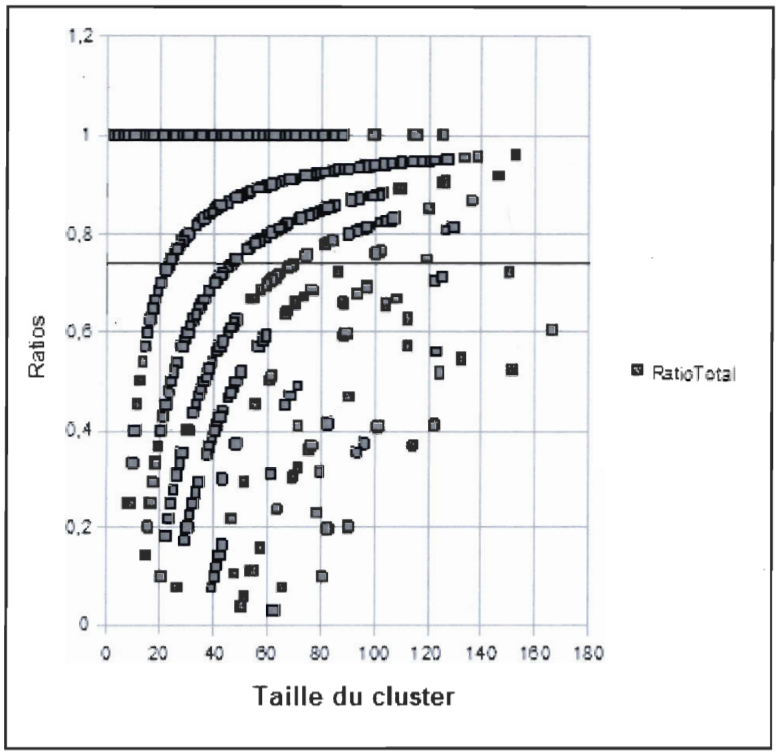


Figure 6.36 Courbe avec 74% de moyenne de cohésion pour 15 mètres

Indicateurs	Valeurs	Pourcentage
Moyenne des tailles des clusters	38	-
Cohésion moyenne	0,74	7%
Nb Total des cellules avant regroupement	2276	100%
Nb Total des cellules regroupées	1191	53,3%
Nb Total de clusters créés après le regroupement	926	-
Nb Total des cellules Libres après regroupement	1085	47,7%

Tableau 6-4 Tableau 6.4 Tableau statistique du regroupement de 15 mètres

Nous constatons que la moyenne de cohésion est de 74%. Elle est légèrement moindre que celle de 10 mètres. La figure 6.38 montre que la moyenne des cellules totales par cluster a augmenté. Elle est égale à 38 cellules. Nous remarquons également que la cohésion diminue par rapport aux distances de 5 et de 10 mètres. Ceci s'explique par l'ajout de cellules NSA dans ces clusters et par conséquent les cellules significatives deviennent plus distantes.

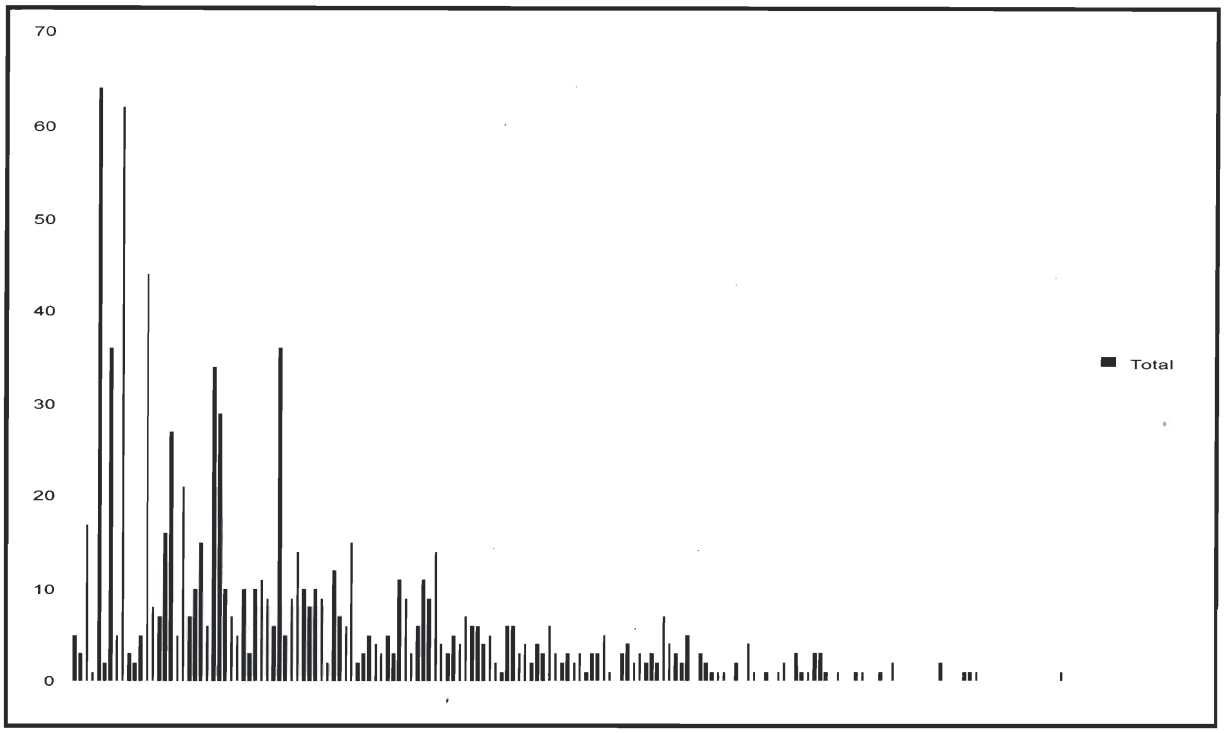


Figure 6.37 Histogramme du nombre de cellules total par cluster de 15 mètres

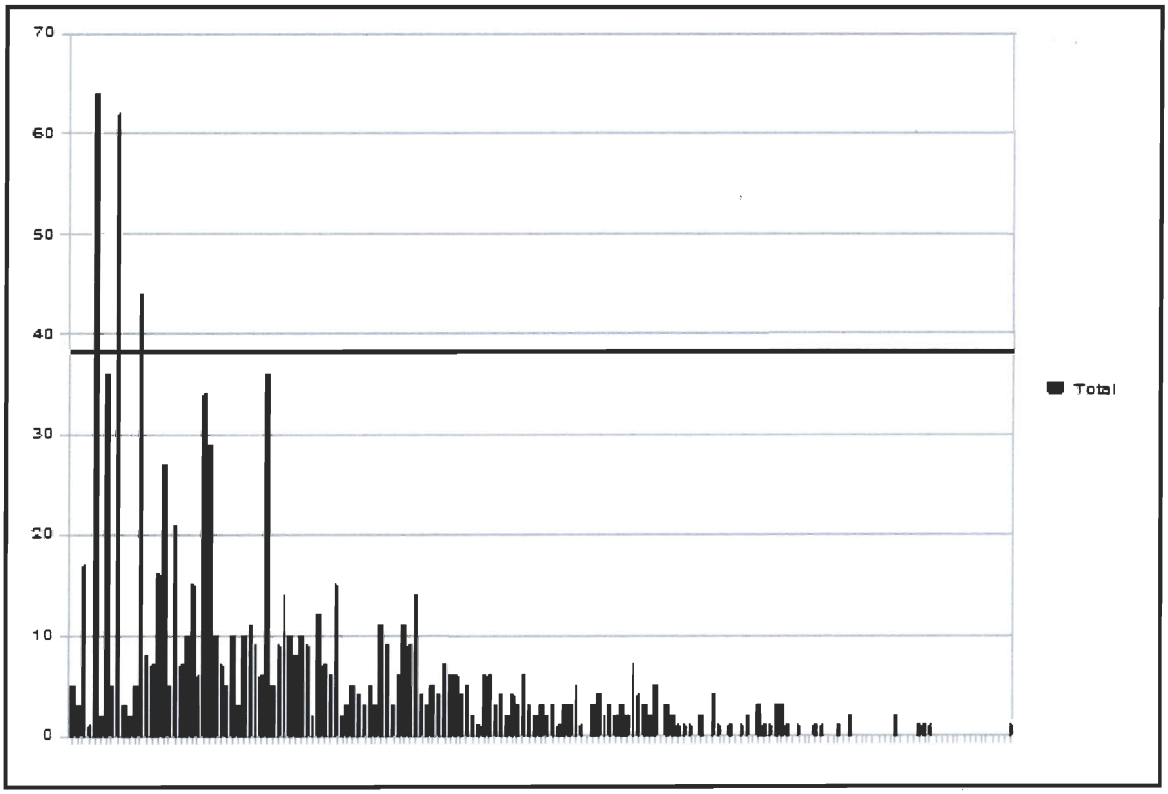


Figure 6.38 Moyenne de 38 cellules total par cluster de 15 mètres

4) Distance de 20 mètres

Les figures suivantes représentent les résultats quantitatifs pour un clustering de 20 mètres

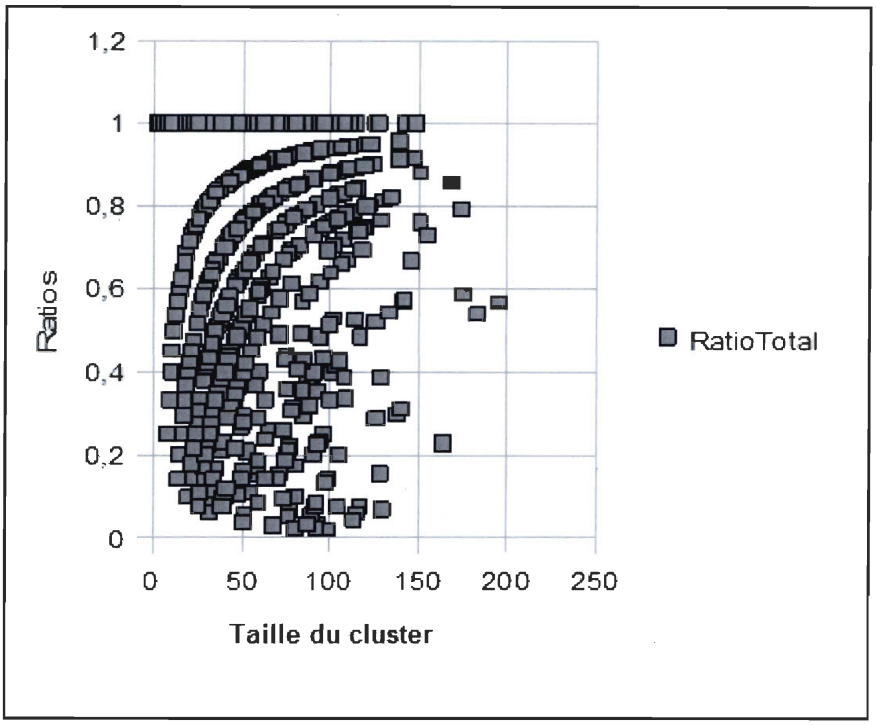


Figure 6.39 Courbe du regroupement d'une distance de 20 mètres

Indicateurs	Valeurs	Pourcentage
Moyenne des des clusters	43	-
Cohésion moyenne	0,68	68%
Nb Total des cellules avant regroupement	2276	100%
Nb Total des cellules regroupées	1325	58,3%
Nb Total de clusters créés après le regroupement	1008	-
Nb Total des cellules libres après regroupement	951	41,7%

Tableau 6-5 Tableau 6.4 Tableau statistique du regroupement de 20 mètres

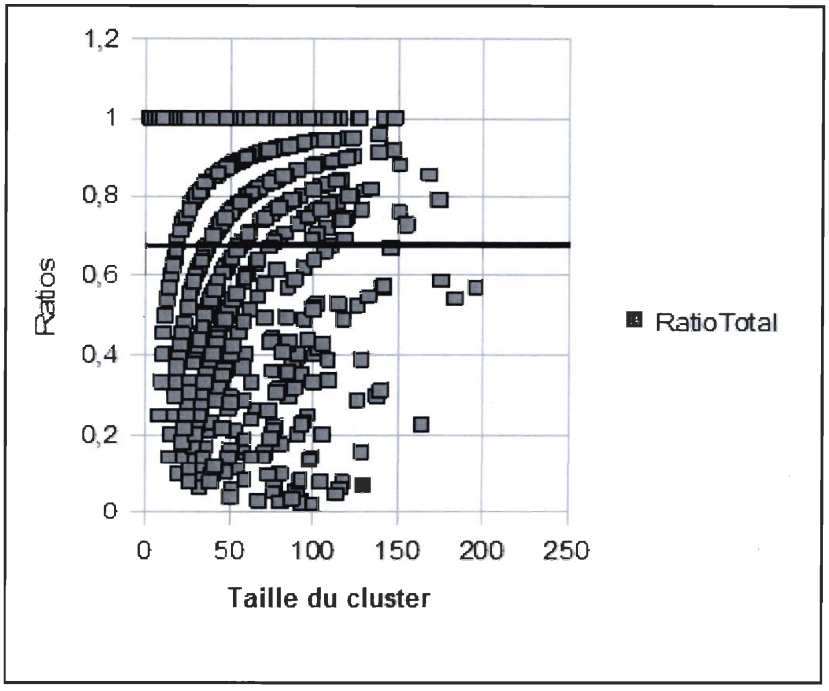


Figure 6.40 Courbe avec 68% de cohésion pour 20 mètres

Pour une distance de 20 mètres, nous remarquons également que le nombre des clusters a encore augmenté (voire figure 6.39). Nous remarquons aussi une augmentation du pourcentage des cellules regroupées. Sa valeur est de 34,4% pour 5 mètres et 46% pour 10 mètres. Ensuite, pour 15 mètres ce pourcentage est égal à 53.3% et il est égal à 58,3% pour 20 mètres. Ceci implique que proportionnellement le nombre de cellules NSA dans chaque cluster a augmenté. Ceci est cohérent avec les résultats de la figure 6.28 où on observe des clusters très larges comparativement à ceux des distances de 5 et de 10. Le nombre de clusters qui sont créés après le regroupement est égal à 1008. Nous remarquons que le nombre de cellules total par cluster a augmenté. Nous concluons que pour cette distance de 20 mètres, la cohésion des clusters est faible.

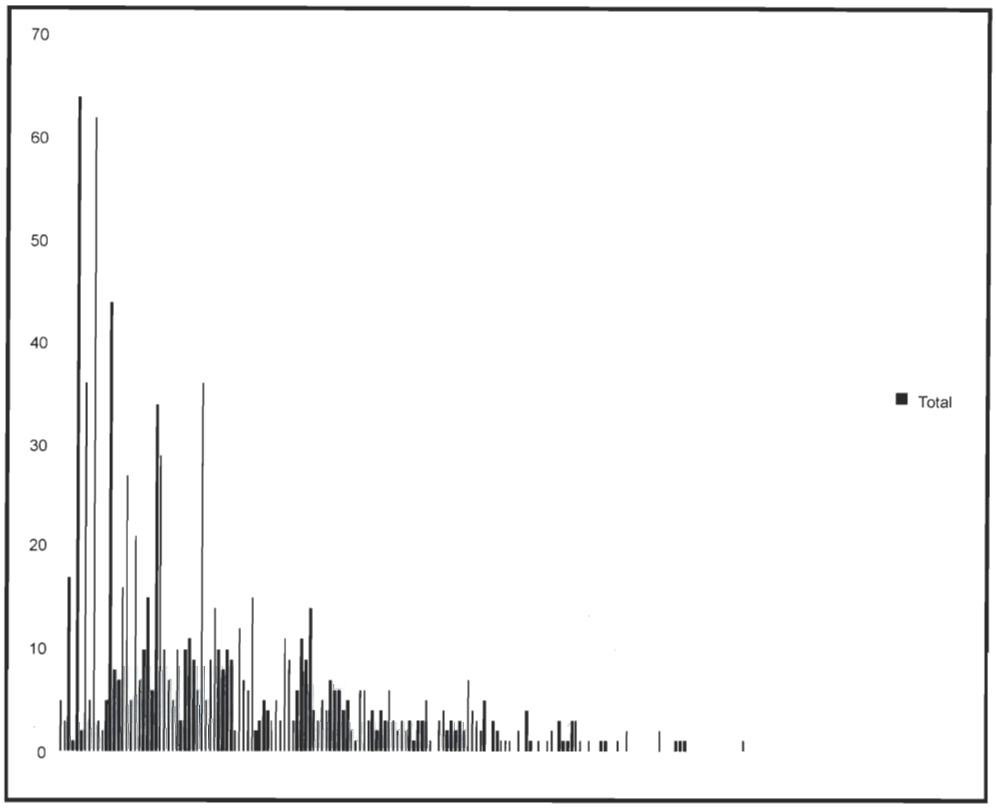


Figure 6.41 Histogramme du nombre de cellules total par cluster de 20 mètres

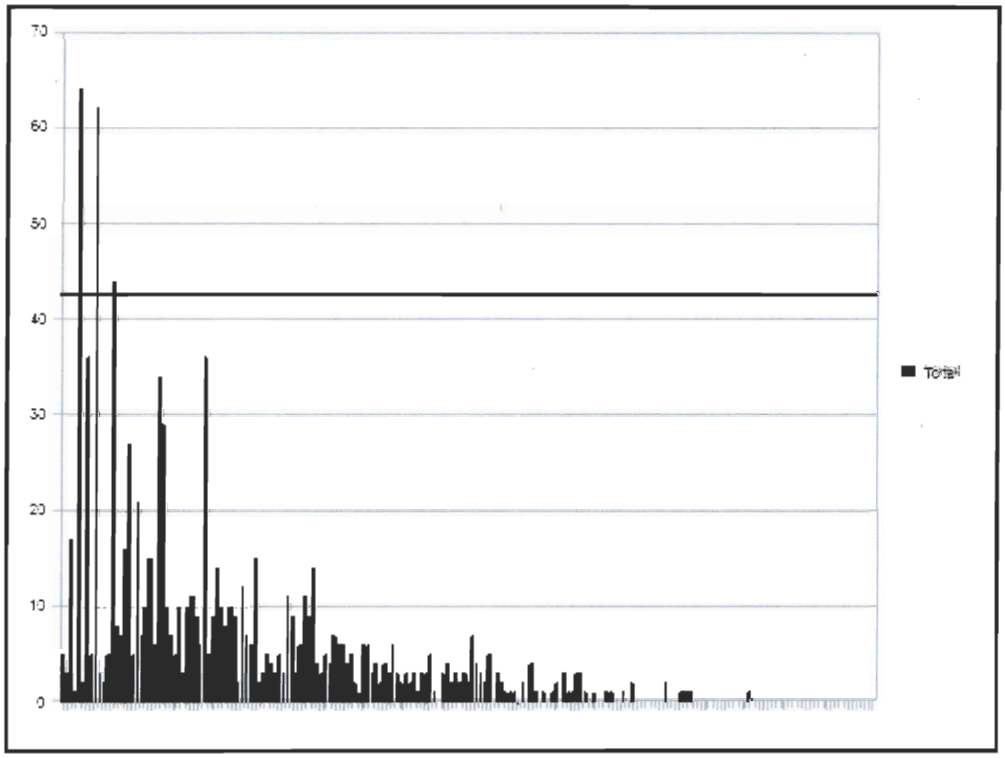


Figure 6.42Moyenne de 43 cellules total par cluster de 20 mètres

6.3.3.3 Conclusion

Nous remarquons que plus on augmente la distance, plus le nombre total de cellules d'un cluster augmente et par conséquent les clusters deviennent plus larges. Ceci diminue la cohésion des clusters dans le sens où les cellules significatives, c'est-à-dire qui possèdent une sémantique sont plus distantes les unes des autres à l'intérieur du cluster, c'est-à-dire il y a plus de cellules NSA dans ce cluster, et par conséquent la distance de similarité entre ces cellules est faible. Nous avons remarqué également, que malgré une faible distance géométrique, on peut trouver un grand nombre de cellules libres qui ne se sont pas regroupées, comme dans le cas d'une distance de 5 mètres, où le pourcentage total de ces cellules est égal à 65,46%. Nous remarquons aussi l'augmentation du nombre des cellules regroupées à chaque distance. Ce nombre évolue de 34% pour 5 mètres à 58% pour 20 mètres. Ceci s'explique par la nature de l'algorithme de regroupement, puisque plus on augmente la distance, plus le nombre de cellule à regrouper augmente. Cependant, le nombre de cellules isolées diminue. Nous remarquons également que la cohésion est forte pour les distances 5 et 10 mètres alors qu'elle est légèrement faible pour celle de 15 mètres et faible pour 20 mètres. Nous concluons que la distance de similarité est plus grande dans les clusters de 5 à 10 mètres, car ces clusters sont plus cohésifs que ceux de 15 et 20 mètres.

Indicateurs	Valeurs 5 mètres		Valeurs 10 mètres		Valeurs 15mètres		Valeurs 20 mètres	
Moyenne des tailles des clusters	24	-	32	-	38	-	43	-
Cohésion moyenne des clusters	0,84	84%	0,76	76%	0,74	74%	0,68	68%
Nb Total des cellules avant regroupement	2276	100%	2276	100%	2276	100%	2276	100%
Nb des cellules regroupées	786	34,36%	1047	46%	1191	53,3%	1325	58,3%
Nb Total de clusters créés après le regroupement	551	-	789	-	926	-	1008	-
Nb Total des cellules libres après regroupement	1490	65,46%	1229	54%	1085	47,7%	951	41,7%

Tableau 6-6 Tableau récapitulatif des valeurs pour chaque distance

6.4 Approche par automates cellulaires

Contrairement à l'approche précédente qui se base sur les distances sémantiques et les distances euclidiennes pour faire du clustering, cette deuxième approche se base sur des

règles affectées à des automates cellulaires. En effet, chaque cellule est dotée de règles qui sont similaires aux règles de voisinages et aux règles de transitions dans les automates géographiques fixes de Benenson (Benenson et al 2005). Ainsi, chaque cellule doit vérifier ces règles pour éventuellement se regrouper avec d'autres cellules de son voisinage. En d'autres termes, chaque cellule déterminera par elle-même, avec quelles autres cellules du voisinage elle se regroupera pour former un cluster. Dans cette section, nous présentons les différents types de règles que nous proposons pour l'automate cellulaire, ainsi que la configuration spatial de son voisinage.

6.4.1 Voisinage et configuration spatiale

Rappelons que Benenson (Benenson et al 2005) présente dans ses travaux les configurations spatiales de Moor et de Von Newman pour un automate cellulaire qui utilise des grilles carrés. Dans notre cas, nous avons plutôt utilisé la configuration classique en fleur qui est utilisée dans les structures *GBT« Generalised Balanced Ternary »* (Gibson, 1982) pour les grilles hexagonales. La figure 6.43 montre une configuration spatiale de notre automate cellulaire hexagonal.

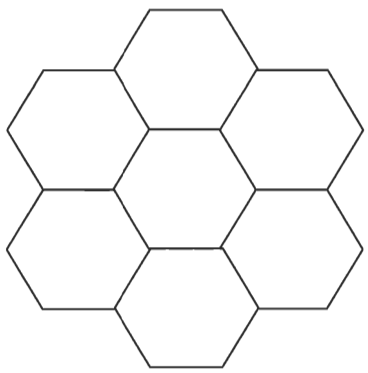


Figure 6.43 Configuration spatiale d'un automate cellulaire hexagonal

Dans cette section nous introduisons la notion de portée de voisinage qui correspond au nombre de cellules à considérer pour le regroupement à partir d'une cellule initiale. En fait, ce n'est autre que la distance entre la cellule centrale et la cellule la plus extrême de son voisinage en termes de nombre de cellules. Ceci est analogue à la notion de rayon dans un cercle, à l'exception qu'un rayon peut être exprimé en mètres, alors que la portée est exprimée par nombre de cellules.

6.4.2 Règles de voisinage

Nous avons repris l'idée d'utiliser des index pour déduire des équations mathématiques qui nous permettront de fixer un voisinage d'une cellule donné. Ceci est conforme à l'approche que Samet (Samet, 1999) a utilisée pour transiter horizontalement et verticalement d'une cellule triangulaire à une autre, sauf que dans notre cas, il s'agit de transitions entres des cellules hexagonales. On définit les règles suivantes. Soit une cellule hexagonale centrale *h* d'index *Id*. Soit *L* le nombre de cellules qui se trouvent sur la largeur de la grille hexagonale à un niveau donné. Soit *m* la portée du voisinage en termes de nombre de cellules par rapport à la cellule *h*.

- La cellule hexagonale voisine d'en haut est telle que son index est : Id+ (m x 1)
- La cellule hexagonale voisine d'en bas est telle que son index est : Id – (m x 1)
- La cellule hexagonale voisine d'en haut à droite est telle que son index est :
  - Id + (m x (L+1))
- La cellule hexagonale voisine d'en bas à droite est telle que son index est :
  - Id + (m x L)
- La cellule hexagonale voisine d'en bas à gauche est telle que son index est :
  - Id – (m x L)
- La cellule hexagonale voisine d'en haut à gauche est telle que son index est :
  - Id – (m x (L+1))

La figure 6.44 montre un exemple de ces règles dans le cas où m = 1.

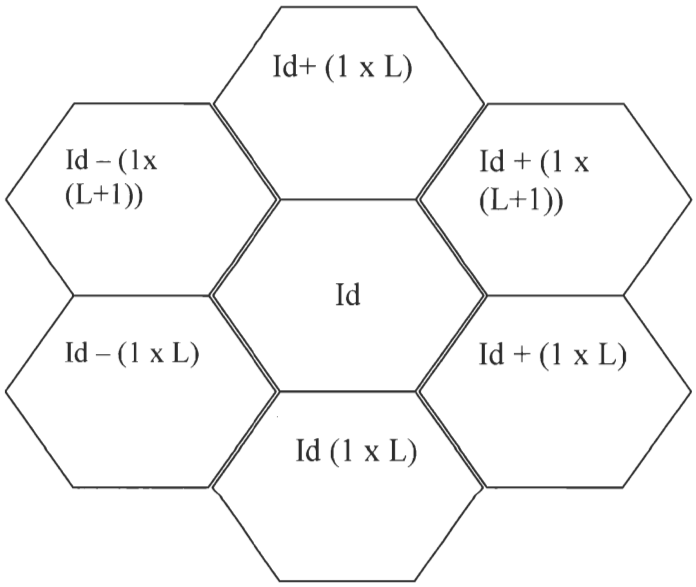


Figure 6.44 Exemple des règles de voisinage pour m = 1

6.4.3 Règles de transition

Tout d'abord, définissons une transition d'états entre cellules dans notre contexte. Une cellule change d'état, si elle peut se regrouper avec une autre cellule de son voisinage. Dans ce cas nous avons une nouvelle cellule émergente. Cette dernière possède à son tour un nouvel état, comme par exemple le nombre de cellules qui lui sont associées ou le nombre de déficit de véhicules de toute cette entité émergente. Ce changement d'état se produit quand une cellule décide de se regrouper avec une autre cellule. Cette prise de décision reflète la capacité d'auto-regroupement de chaque cellule à l'aide d'une règle de décision. Cette dernière est équivalente à la condition sémantique dans l'approche précédente. Ainsi, le changement d'état reflète le regroupement entre cellules et se produit si une règle de décision de la cellule considérée est satisfaite. Dans ce cas, les règles de transition sont exécutées. Cette règle de décision est la suivante :

$$\textbf{Si } P_{c_i} > P_{c_j}, \forall\ j \in V\text{, alors Exécution des règles de transitions}$$

Pci et *Pcj* représentent les potentiels ou les valeurs sémantiques qui sont associés aux cellules ou aux clusters C d'indice *i* et *j*. *V* est le voisinage de ces cellules ou de ces clusters.

6.4.4 Algorithme de regroupement des cellules

Dans cette approche, nous avons aussi utilisé deux étapes; une pour l'initialisation et une autre pour le regroupement en tant que tel. Dans cette section, nous présentons les algorithmes d'initialisation et de regroupement.

6.4.4.1 Initialisation

Lors de cette étape, nous avons gardé le même algorithme que nous avons utilisé dans l'approche de clusterisation. Nous obtenons comme résultat de cet algorithme la liste *ListHex* qui contient les éléments triangulaires qui sont localisés sur la carte. Elle est transmise comme paramètre à l'algorithme responsable du regroupement selon une approche par automates cellulaires. Nous présentons cet algorithme et ses fonctions dans la section qui suit.

6.4.4.2 Regroupement

L'algorithme principal de regroupement a les paramètres suivants:

- *ListHex* : La liste qui représente les premiers clusters qui sont créés lors de l'étape d'initialisation
- *Couleur1* et *couleur 2* : deux couleurs différentes pour dessiner les clusters de forte et de faible densité.
- *Nbvoisins* : le nombre de cellules distantes d'une cellule centrale. Il peut être considéré comme le rayon d'un voisinage en termes de cellules.

Cet algorithme parcourt la liste des premiers clusters *ListHex.* Ensuite, il fait appel à trois autres fonctions qui appliquent les règles de voisinage, de décision et de transition pour chaque cluster de cette liste. En fait, chaque cluster *H* qui est obtenu de la liste *ListHex* est transmis comme paramètre à la fonction *RèglesVoisinage.* Cette dernière détermine le voisinage *V* du cluster *H.* Ensuite, pour chaque élément *H2* de *V*, la fonction *RèglesDécison* spécifie si les clusters *H* et *H2* peuvent se regrouper. Le résultat est une valeur booléenne *reg.* La valeur « vraie » de cette dernière permettra d'exécuter les règles de transition entre les deux cellules (voire figure 6.45).

```
Automates ( List ListHex, Couleur1,Couleur2, NbVoisin)
Pour chaque cellule hexagonale H dans ListHex
Liste V= RèglesVoisinages(H.NbVoisin)          // Règles de voisinage
Pour Chaque Hexagone H2 dans V
Booléen reg = RèglesDecision( H1,H2)           // Règles de décision
 Si (reg = vrai)
  RèglesTransition(H1,H2, Couleur1,Couleur2)   // Règles de transitions
 Fin Si
Fin
```

Figure 6.45 Algorithme principal de regroupement par automates cellulaires

Concernant la fonction *RèglesVoisinage,* elle permet de déterminer le voisinage d'un hexagone H. En effet, ce dernier est passé comme paramètre à cette fonction qui déterminera alors son index dans la grille hexagonale. Ainsi, on trouve l'index *Id* de cet hexagone et on le considère comme l'index de la cellule centrale, c'est-à-dire la cellule à partir de laquelle nous commençons à chercher le voisinage. Finalement, cette fonction exécute les règles de voisinage que nous avons définies plus haut et elle retourne la liste des hexagones qui se trouvent dans un voisinage de la cellule hexagonale centrale d'index Id, et par le fait même cela constitue le voisinage de la cellule centrale. La figure 6.46 montre cette fonction.

```
RèglesVoisinages(Hexagone H)
Id← Obtenir l'index de H dans la grille     // l'index de la cellule centrale
ListeH← Initialiser la liste
Pour  m = 1; m <= NbVoisin; m++
                Hexagone H1 ← Grille [id + (m * 1)]
                Hexagone H2 ← Grille [id - (m * 1)]
                Hexagone H3 ← Grille [id + (m * L)]
                Hexagone H4 ← Grille [id + (m * L+1)]
          Si ( id>=L )                  // vérifier si la soustraction est possible
                Hexagone H5 ← Grille [id - (m * L)]
                Hexagone H6 ← Grille [id - (m * L+1)]
                Ajouter H5 H6  à la ListeH
            FinSi
           Ajouter H1, H2 H3 H4 à la ListeH
FinPour
 Retourner ListeH
```

Figure 6.46 Algorithme des règles de voisinage

Nous avons mentionné la fonction *RèglesDécison* quand nous avons présenté l'algorithme principal de regroupement. Cette dernière n'est qu'une simple comparaison entre les moyennes des valeurs sémantique de chaque hexagone. Son but est de comparer deux hexagones et de renvoyer une valeur booléenne égale à vrai si un des deux hexagones a une moyenne des valeurs sémantique qui est supérieure ou égale à celle de l'autre. Elle renvoie une valeur booléenne égale à faux dans le cas échéant. Cette fonction fait appel à une autre fonction *Moyenne* qui est responsable du calcul proprement dit. Les figures 6.47 et 6.48 présentent ces deux fonctions.

```
RèglesDécison (Hexagone H1, Hexagone H2)
Réel M1 = MoyenneValSemantique(H1)          // Obtenir la moyenne de H1
Réel M2 = MoyenneValSemantique (H2)         // Obtenir la moyenne de H2
Si M1 >= M2  Alors retourner VRAI Sinon retourner FAUX
Fin
```

Figure 6.47 Algorithme des règles de décision

```
Réel Moyenne (Hexagone H)
Pour chaque Triangle T dans H
Somme + =T.Valeur;  // calcule de la somme
nombreTriangle++    // incrément du nombre de triangle
Fin Pour
Retourner  (Somme / nombreTriangle )
```

Figure 6.48 Algorithme de la fonction de calcul de la moyenne d'un hexagone

Finalement, la fonction *RèglesTransition*, est similaire à la fonction *Copy* dans l'approche de clustering. Elle admet comme paramètres les deux hexagones d'intérêt qui sont spécifiés par la fonction précédente. Elle admet aussi les deux couleurs qui sont transmises à l'algorithme principale et qui sont propagées jusqu'à cette fonction. Cette dernière permet de changer l'état des deux clusters qui sont identifiés par la fonction *RèglesDécision.* Ainsi, elle permet que les deux hexagones se regroupent afin de créer un nouveau cluster émergent. Cette fonction calcule de nouveau la moyenne globale de l'entité émergente. Ensuite, elle ajoute l'hexagone absorbé à la liste des éléments appartenant à l'hexagone qui avait la valeur sémantique la plus élevée. Finalement, elle dessine ce dernier avec une

couleur foncée qui est *couleur1* et elle dessine également l'hexagone qui avait une faible valeur sémantique avec la couleur *couleur2* qui est pâle. Cette fonction est présentée par la figure 6.49.

```
RèglesTransition(Hexagone H1,Hexagone H2, couleur1, couleur2)
Réel m2 ← Moyenne(H2)
H1.m1 = (H1.m1 +m2)/2
Ajouter H2 à H1.listehexagone
Dessiner H1 avec couleur1
Dessiner H2 avec couleur2
Fin
```

Figure 6.49 Algorithme des règles de transitions

6.4.5 Résultats obtenus

Dans cette section, nous présentons les résultats que nous avons obtenus.

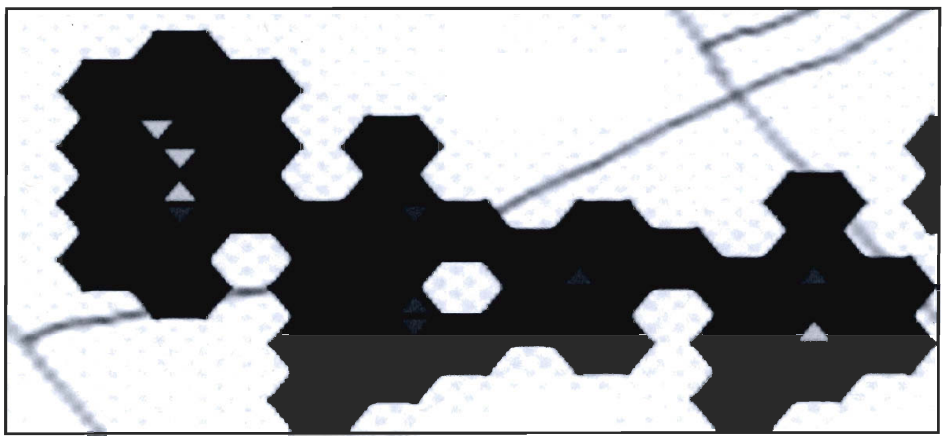


Figure 6.50 Vue élargie des clusters obtenus par automates cellulaires

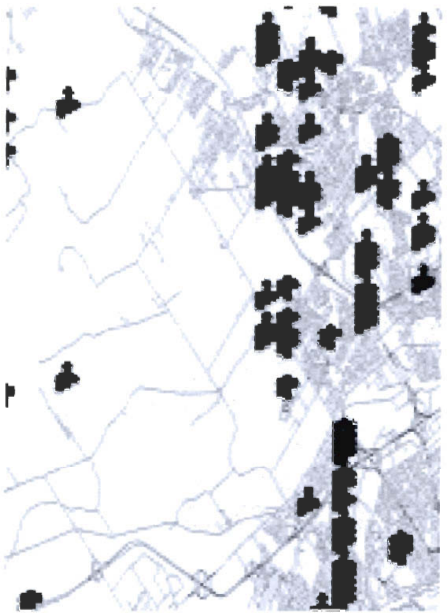


Figure 6.51 Vue d'ensemble des clusters obtenus par automates cellulaires

Ces résultats représentent des clusters qui ont une couleur noire et qui sont formés par des cellules hexagonales comme le montre les figures 6.50 et 6.51. Ces clusters sont regroupés selon des propriétés sémantiques communes. Dans cet exemple, il s'agit du nombre d'adolescents par ménage en 2002. Les cellules qui ont une valeur sémantique élevée sont colorées en bleu foncé. Celles qui ont une valeur sémantique faible sont colorées en bleu pâle. Lors d'un deuxième test, nous avons changé les couleurs qui désignent les cellules avec une valeur sémantique élevée et faible. Lors de ce test, nous nous sommes intéressés à la dimension sémantique du déficit en voitures, à l'image du test que nous avons effectué avec l'approche par clustering. Nous avons utilisé également la couleur rouge pour une valeur sémantique élevée et la couleur jaune pâle pour une faible valeur sémantique. Nous avons choisi ce contraste de couleurs pour fournir une métaphore visuelle significative pour l'usager. Ainsi, il pourra distinguer facilement les cellules avec des valeurs sémantiques faibles et élevées. La portée de voisinage est fixée à 1 dans ce cas. Les figures 6.52 et 6.53 présentent ces résultats. Ensuite, nous avons modifié le paramètre de portée de sorte à ce qu'il soit égal à deux. Nous obtenons les résultats qui sont présentés dans les figures 6.54 et 6.55. Finalement, on a fixé la portée à 3 cellules. Nous obtenons les résultats des figures 6.56a et 6.57.

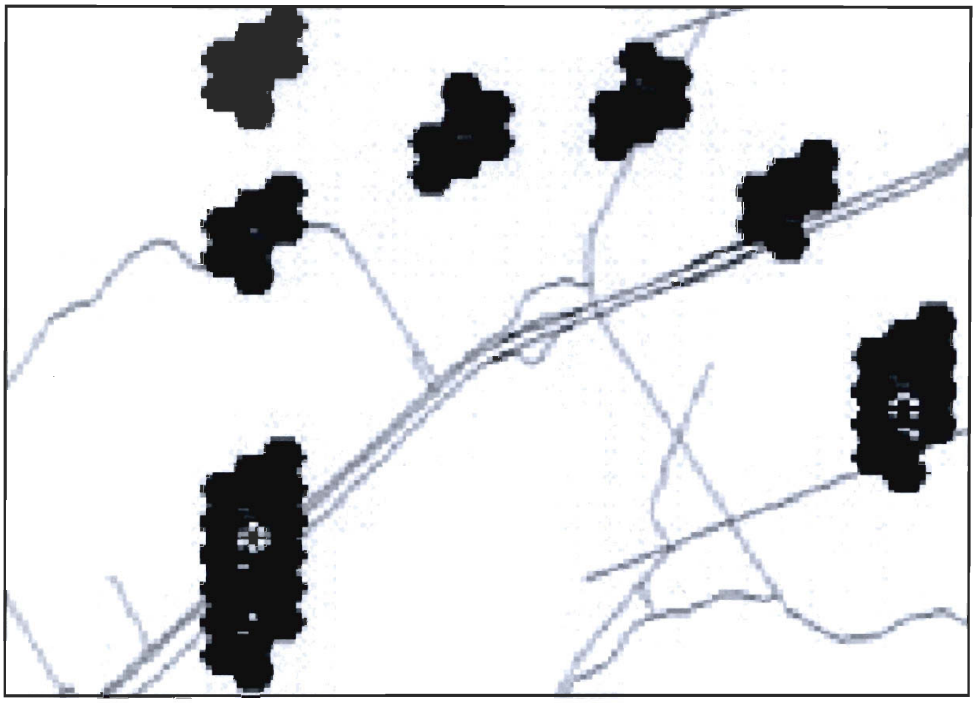


Figure 6.52 Vue partielle des zones de déficit en voitures d'une portée d'une cellule

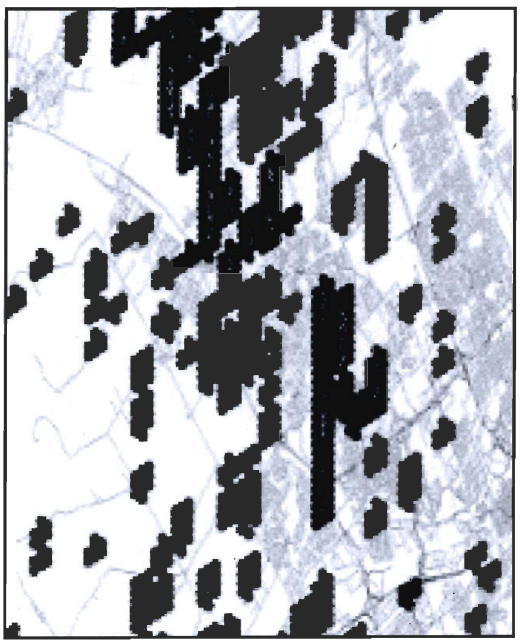


Figure 6.53 Vue d'ensemble des zones de déficit en voiture d'une cellule de portée

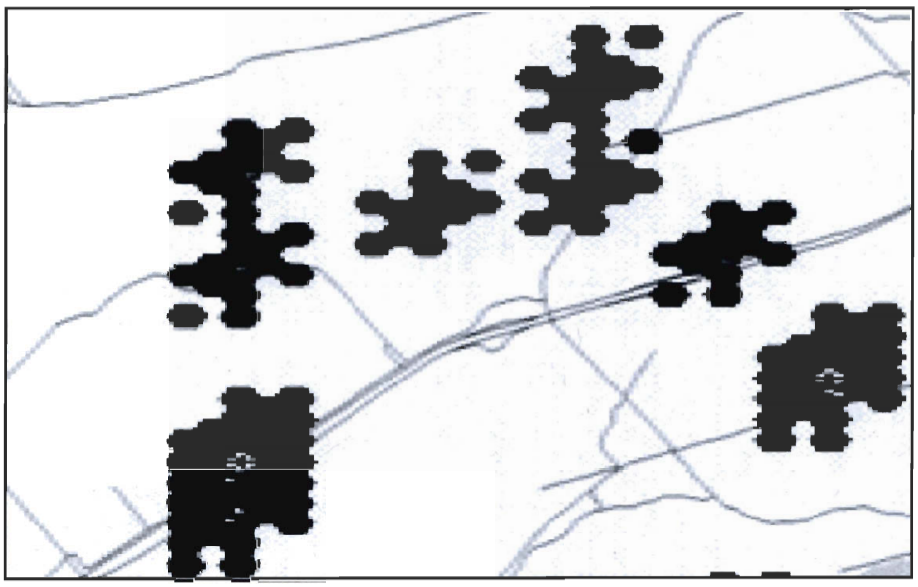


Figure 6.54 Vue élargie des zones de déficit de 2 cellules de portée

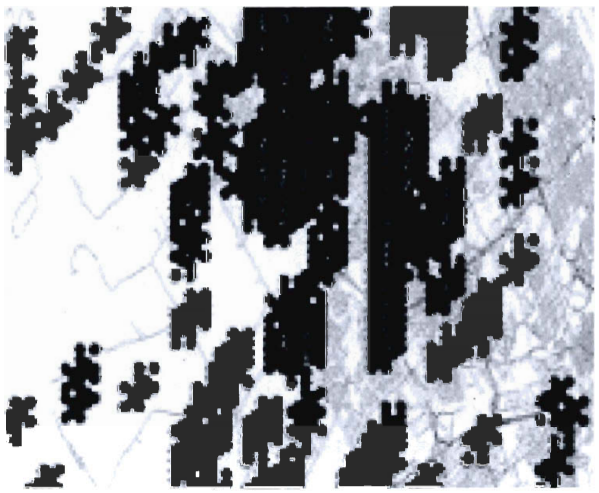


Figure 6.55Vue d'ensemble des zones de déficit en voitures de 2 cellules de portée

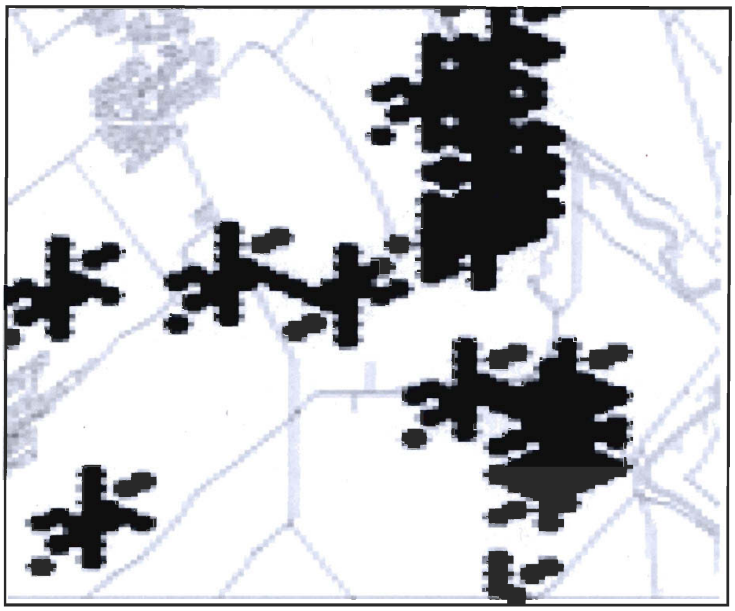


Figure 6.56 Vue élargie des zones de déficit en voitures de 3 cellules de portée

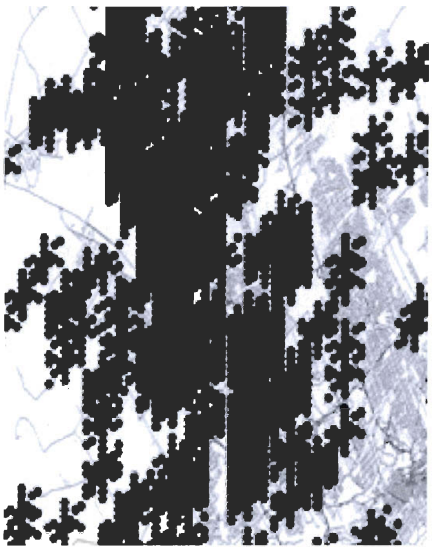


Figure 6.57 Vue d'ensemble des zones de déficit en voitures de 3 cellules de portée

Nous remarquons qu'au-delà d'une portée de trois cellules les clusters deviennent très larges et les cellules d'intérêts deviennent distantes. Ceci est similaire aux résultats que nous avons obtenus dans la section 6.3.3 où nous trouvions que la distance seuil du regroupement pour l'approche par clustering est de 20 mètres.

6.5 Conclusion

Dans ce chapitre nous avons présenté l'architecture de notre application et son environnement technologique. Nous avons présenté également les différentes classes et packages qui la constituent. Nous avons utilisé une méthodologie par prototypage et une conception basée sur les patrons de conception pour créer cette application. Dans une deuxième partie, nous avons introduit la notion de regroupement (Clustering) et les techniques que nous avons utilisées. Nous nous sommes inspirés de deux approches différentes, mais toutes les deux issues du domaine de l'intelligence artificielle. La première est basée sur le clustering, plus particulièrement sur l'algorithme de clustering par fusion. La deuxième approche est basée sur les automates cellulaires. En effet, nous avons réussi à regrouper des cellules hexagonales par l'intermédiaire d'un mécanisme de règles. Les entités émergentes dans les deux cas fournissent des cartes de densités qui représentent des zones ayant des propriétés sémantiques communes. Par ailleurs, après avoir présenté les résultats des algorithmes de regroupements des deux approches, dans la section suivante nous allons discuter ces résultats, mentionner les travaux futurs et finalement conclure de manière générale par rapport aux objectifs qui étaient visés par ce projet.

Chapitre 7 Conclusion

7.1 Discussion

Dans cette section nous allons discuter les résultats que nous avons présentés dans le Chapitre 6 et ceux que nous avons présentés dans le chapitre 5. Nous allons également présenter une conclusion générale pour l'ensemble de notre projet et évoquer les étapes futures de ce travail.

7.1.1 Approche de clustering versus approche par automates cellulaires

Dans l'approche de clustering, l'algorithme employé est celui par fusion. En utilisant les heuristiques que nous avons présentées dans la section 6.3.2.1, nous avons amélioré la répartition initiale des données par rapport à son utilisation traditionnelle, où on emploie généralement une répartition aléatoire. Notre approche permet de faire du clustering qui se base sur des distances de similarités sémantiques et des distances géométriques. Dans le cas de l'approche par automate cellulaire, nous avons utilisé des règles. Nous avons trouvé une ressemblance entre les deux approches. En effet, la distance de similarité sémantique dans l'approche de clustering est semblable aux règles de décisions. Quant à l'algorithme de regroupement, il est similaire aux règles de transitions qui activeront le regroupement au niveau des automates cellulaires. Finalement, les règles de voisinage sont semblables aux distances géométriques sauf que la distance au lieu d'être euclidienne et exprimée en termes de nombre de cellules pour les automates. Nous avons gardé les mêmes heuristiques au niveau de l'initialisation des deux approches. Par contre, nous avons introduit la notion de voisinage qui est propre aux automates cellulaires et qui est absente au niveau du clustering.

Par ailleurs, nous avons comparé les deux approches en termes de performances, c'est-à-dire celle qui est la plus rapide à s'exécuter. Nous avons trouvé que l'approche par clustering est moins rapide que celle basée sur les automates cellulaires. En effet, l'approche par clustering ne fait que comparer des valeurs et de calculer des distances entre deux cellules à la fois. Ce calcul est imbriqué dans trois boucles répétitives. La complexité de l'algorithme est cubique. Par contre pour l'approche par automates cellulaires, le calcul

des distances géométriques est similaire aux règles de voisinage. Certes, pour une cellule donnée, au lieu de faire des calculs de distances géométriques pour déterminer les cellules voisines, on utilise des équations mathématiques (voir chapitre 6) sur les index des cellules. Plus le nombre des cellules du voisinage est grand, plus cette opération est lourde puisque le nombre de cellules à examiner augmente. L'algorithme de cette opération est O(n). Ainsi, la complexité totale de l'algorithme de regroupement est carrée. De plus, l'opération de calcul des index des cellules voisines n'est autre qu'une équation d'addition. Dans le cas de l'approche de clustering, il s'agit de calcul géométrique et par conséquent, il est moins rapide.

7.1.2 Approches intelligentes versus l'approche d'analyse spatiale

Dans cette section nous discutons l'apport des techniques intelligentes par rapport à la technique conventionnelle d'analyse spatiale. Tout d'abord, les résultats obtenus par les approches de clustering et d'automates cellulaires sont similaires à des cartes de densités dans les SIG et que nous avons présentées dans le chapitre 5. En fait, nous avons choisi des couleurs foncées et pâles pour offrir une métaphore visuelle facile à interpréter par l'usager. Pour l'approche de l'analyse spatiale, on observe des cellules qui sont éparpillées dans l'espace et chacune avec un code couleur. Il n'y a aucun lien entre ces cellules et par conséquent, elles ne forment pas des clusters. Toutefois, on peut observer à l'œil nu des regroupements de cellules. C'est l'usager qui interprète ce résultat et il n'y a pas de mécanisme au niveau des outils SIG qui peuvent donner des statistiques ou des opérations spatiales sur le cluster observé visuellement. De plus, l'usager peut analyser les données, créer des cartes, mais c'est à lui d'interpréter les résultats obtenus. Il est obligé d'utiliser des requêtes SQL pour effectuer son analyse. Ceci complexifie la tâche des décideurs vu que ça leur demande une connaissance des outils. De plus, l'usager n'a pas un contrôle sur le processus de regroupement. Nous avons vu dans le chapitre 5 que les cartes sont créées par l'entremise de requêtes spatiales propre au logiciel MapInfo. Comme il existe de nombreux logiciel de ce genre, chacun implémente ses requêtes spatiales de manière différente. Ceci rajoute un autre élément de complexité à ce genre d'étude, dans le sens où l'usager est obligé de se familiariser avec les requêtes propres au SIG en question. Par contre, avec notre approche, l'usager n'a pas besoin d'avoir des connaissances informatiques. Ce genre d'outil s'adresse à des décideurs qui ne connaissent pas SQL. En

effet, à l'aide des techniques de clustering et d'automates cellulaires, l'outil permet de faire d'une part de l'analyse spatiale puisqu'il permet de répartir les données sur la carte, localiser les cellules triangulaires d'intérêt et affecter des codes de couleurs à ces dernières et d'autres part d'offrir des méthodes avancées pour analyser les données. Le plus important, et c'est là où réside son avantage par rapport à l'analyse spatiale, c'est qu'à partir de ces cellules localisées, il permet de créer des clusters. Chaque cluster est une entité à part, qui a ses propres cellules et ses propres valeurs. D'un point de vue visuel, ils sont remarquables directement par l'usager et il n'a pas à faire un effort cognitif pour repérer ces clusters.

7.2 Conclusion

Dans le cadre du projet *MUSCAMAGS*, nous avons utilisé des grilles hexagonales hiérarchiques afin d'analyser des données issues d'une population synthétique représentative de la ville de Québec. Nous avons établi des cartes de densités et nous avons remarqué à l'œil nu des regroupements de cellules. Nous avons essayé de recréer l'émergence de ces formes avec notre prototype afin d'obtenir des regroupements «géo-sémantiques », c'est-à-dire, à la fois d'ordre sémantique et géométrique. Dans cette section nous présenterons les étapes de notre travail et surtout nos réalisations pour atteindre nos objectifs de départ. Nous aborderons également les perspectives de notre travail dans une autre section. Rappelons également les objectifs de cette recherche. Nous avons fixé les objectifs suivants à atteindre.

- Palier aux limites des grilles hexagonales qui sont utilisées dans le projet *MUSCAMAGS* en travaillant sur des grilles dont l'unité géométrique de base est un triangle équilatéral.
- Avoir un mécanisme de construction générique de ce genre de grilles, c'est-à-dire pouvoir paramétrer ces grilles selon l'usage et le contexte de l'application.
- Préparer un jeu de données propice à l'étude d'un phénomène urbain d'intérêt et ce, à partir des données de la population synthétique du projet *MUSCAMAGS*.
- Assigner des automates cellulaires aux grilles hiérarchiques dans le but de créer un mécanisme de regroupement intelligent pour des fins d'exploration des

connaissances, l'analyse spatiale des données et de visualisation de phénomènes émergeants.

- Avoir une interface usager qui répond bien au contexte d'utilisation et qui fournit un mécanisme de visualisation naturel de la hiérarchie à l'aide de la technique de « pan/zoom ».

7.2.1 Conclusion générale

Tout d'abord, au niveau du partitionnement de l'espace, nous avons utilisé un algorithme récursif afin de créer des grilles triangulaires hiérarchiques. Ce procédé se fait selon une approche « top-down ». En effet, partant de triangles de grande taille, on les décompose en des sous-triangles de tailles moindres. Contrairement à l'approche « bottom-up » que nous avons explorée au départ, cette approche assure une généricité au niveau de la structure obtenue. Effectivement, on peut la paramétrer selon différents critères tels que la taille des cellules et le nombre de niveaux. Ainsi, on pourrait facilement l'adapter à d'autres problématiques. Ensuite, à partir de ces cellules triangulaires, nous avons créé des cellules hexagonales hiérarchiques en s'assurant qu'elles ne se chevauchent pas et qu'elles aient un indexage selon le Colomn-ordering, car c'est celui qui est utilisé par la grille hexagonale originelle développée par le CRAD.

Le mécanisme d'indexage que nous avons établi représente un lien entre les différentes cellules hexagonales et triangulaires. Il permet donc de naviguer d'une cellule à une autre de manière, soit verticale (ou hiérarchique) entre les cellules de différents niveaux, soit horizontale, c'est-à-dire entre des cellules d'un même niveau. En ce sens, chaque cellule hexagonale ne contient que des pointeurs vers les cellules triangulaires. Cette méthode permet de ne pas dupliquer l'information et par le fait même de ne pas surcharger la structure hexagonale. Le mécanisme d'indexage que nous avons utilisé ressemble à celui des SIG, c'est-à-dire, qu'il se représenté sous la forme de tables de base de données. Nous avons créé deux types d'index. Un premier que nous avons appelé *index externe*, pour lier les différents hexagones. Un deuxième que nous avons nommé *index interne* qui assure le lien entre les différentes cellules triangulaires. Par conséquent, on peut facilement tracer les items d'un hexagone donné avec des requêtes SQL, d'où sa ressemblance avec les index des SIG (voir chapitre 4).

Dans une deuxième étape, nous avons établi des cartes de densités en se basant sur l'analyse spatiale des données pour constater les avantages d'une telle approche d'un point de vue technique, c'est-à-dire l'efficacité d'utilisation et l'effort cognitif effectué par l'usager afin d'interpréter ce genre de cartes (voir chapitre 5). Finalement nous avons raffiné notre prototype pour qu'il réponde aux objectifs de regroupement « géo-sémantique ». Nous avons utilisé un algorithme de clustering par fusion qui est similaire à l'algotithme du K-means (voir chapitre 6) comme algorithme principal de l'approche de clustering. L'inconvénient de cet algorithme est sa phase d'initialisation, puisqu'il génère les premiers clusters aléatoirement. Pour initialiser cet algorithme, nous avons utilisé deux heuristiques pour augmenter les performances et par le fait même localiser et visualiser les cellules d'intérêts. Nous avons considéré comme premiers clusters les cellules triangulaires qui répondent à une dimension sémantique choisie et qui appartiennent au même hexagone. Ensuite, nous avons calculé le centroïde de chaque polygone ou de chaque cluster et nous avons déterminé des conditions de regroupement sémantiques en définissant une distance sémantique entre deux cellules (comparaison entre les valeurs sémantiques associées aux cellules) et une distance euclidienne (le plus proche voisin). La satisfaction des deux conditions permet un regroupement et par conséquent la mise à jour du centroïde et de la valeur sémantique totale du cluster émergent. Les résultats obtenus sont les clusters émergents. Également, nous avons assigné à chaque cellule de la grille un automate cellulaire. En effet, chaque automate cellulaire sert à trouver les autres cellules avec lesquelles il se regroupera. On parle alors d'auto -regroupement. Le résultat de ce processus est des clusters émergent qui ont une forme spatiale semblable à la structure *GBT* de Gibson (Gibson, 1983). D'un point de vue technique, nous avons utilisé une architecture en couches, qui peut facilement se généraliser en framework. Nous avons également utilisé les patrons de conception et le pattern *MVC* comme une technique de modélisation pour avoir une architecture flexible et maintenable. Quant aux interfaces utilisateurs, elles sont zoomables et utilisent des méthodes appropriées pour faciliter l'interaction avec l'usager. Ils permettent notamment à l'utilisateur de faire du pan et du zoom. Elles offrent aussi un *bird's eye* pour que l'usager ne soit pas désorienté et pour qu'il puisse se situer et naviguer dans l'environnement spatial. La métaphore visuelle utilisée se base sur des codes de couleurs pour que l'usager réussisse à distinguer les différents variables. Par exemple, on

utilise une couleur foncée pour une forte valeur, et une couleur plus pâle pour une valeur moindre. Dans la section suivante, nous présentons les perspectives ouvertes par notre travail.

7.2.2 Travaux futurs

Dans ce qui suit nous rappelons quelques réalisations de notre travail. Au sujet de l'indexage : nous nous sommes basés sur l'approche de Samet (Samet,1990) afin de partitionner l'espace en des cellules triangulaires. Pour indexer les cellules triangulaires, Samet utilise un couple de bits binaires qui sont enregistrés dans une structure de données. Samet accède à ces index par l'intermédiaire d'opérations mathématiques d'additions et de soustractions. Notre mécanisme d'indexage ne ressemble pas à celui de Samet, car il est inspiré des SIG, sous la forme de tables de bases de données. Ceci permet de requêter rapidement les index des cellules en utilisant *SQL*. Cet index établit un lien entre les cellules de la grille originelle dans le SIG *MapInfo*, et les cellules de notre grille hexagonale. Ceci dit, les cellules pourront implémenter un autre index sous la forme d'un arbre B, qui enregistre les « path array » binaires de chaque cellule. Par conséquent, on pourrait utiliser deux sortes d'index: Un index pour indexer les cellules de la grille originelle et un autre sous la forme d'arbre B pour indexer les cellules et les opérations internes de la grille. Cependant, d'autres fonctionnalités sont nécessaires pour assurer le passage d'un index à un autre.

Au-delà de l'approche par clustering, nous nous sommes orientés vers une autre approche qui est basée sur les automates cellulaires. En effet, dans notre prototype chaque cellule peut devenir une entité autonome régie par des règles de transition et par son voisinage. L'intérêt d'utiliser une telle approche est de faire du regroupement par automates cellulaires. Ainsi, chaque cellule, qu'elle soit triangulaire ou hexagonale, décidera, par l'entremisse de ses règles de voisinage, de décision et de transition, avec quelles autres cellules elle peut se regrouper. Ainsi, on peut observer des entités qui émergent et qui représentent le résultat du regroupement entre plusieurs cellules. Nous pouvons également inclure des outils pour déterminer la qualité du regroupement en fournissant un mécanisme d'analyse quantitative des résultats obtenus comme nous l'avons effectué dans la section 6.3.3 du chapitre 6. En effet, grâce aux composantes COM/COM+ de la plateforme .NET,

nous pouvons inclure des fonctionnalités pour générer des diagrammes statistiques qui reflètent la qualité du regroupement et permettraient au décideur d'avoir une meilleure idée sur les paramètres d'entrée du mécanisme de regroupement. Rappelons que d'autres techniques de data Mining peuvent être rajoutées par exemple à partir des clusters dispersés dans l'espace, nous pouvons utiliser des règles d'associations ou des techniques de classification. Nous pouvons également utiliser des techniques de visualisations 3D, comme l'extrusion. En effet, on peut définir un axe (x,y,z) et élever les coordonnées de chaque cellule de coordonnées (x,y) selon l'axe des z. Ainsi, on aurait des cellules triangulaires ou hexagonales en 3 dimensions. On peut corréler la hauteur de l'élévation selon l'axe des z avec le potentiel de chaque cellule afin que l'usager puisse remarquer facilement la différence de potentiel entre les cellules d'un même cluster. On pourrait également, utiliser une structure d'Octrees pour enregistrer les différentes cellules à la suite d'un partitionnement de l'espace. Aussi, on pourrait remplacer la structure régulière par une structure irrégulière hiérarchique. Ainsi ce travail ouvre de nombreuses perspectives de recherche passionnantes.

Bibliographie

Bell, J 1983. *Spatially referenced methods of processing raster and vector data*, Image Vision Computing, Vol. 1(4), pp. 211-220.

Benenson I et Torrens, P. 2005. *A minimal prototype for integrating GIS and geographic simulation through Geographic Automata Systems. In GeoDynamics*, ed. F. Wu, pp 92.

Bentley, J.L. 1975. *A survey of techniques for fixed radius near neighbour searching*. SLAC Rep. No. 186, Stanford Linear Accelerator Center, Stanford University, Stanford, Calif., Aug.

Bentley, J. L., and Stanat D. F., 1975. *Analysis of range searches in quad trees*. Inf. Process. Lett. 3, 6 (July), pp. 170-173.

Bentley, J. L., and Friedman, J. H., 1979. *Data structures for range searching*. ACM Comput. Surv. *11,* 4 (Dec.), pp. 397-409.

Burt P.J 1980. *Tree and pyramid structures for coding hexagonally sampled binary images*. Computer Graphics and Image Processing, Vol 14, pp .271-280.

Chaker Walid 2009. *Vers la géosimulation multi-agents et multi-échelles*.

Rousseau C, 2007. *Mathamatiques et technologie*. Springer, 2007, pp. 43.

Dutton. G., 1990. *Locational properties of quaternary triangular meshes*. In Proceedings of the Fourth International Symposium on Spatial Data Handling, pp. 901–910.

Dutton. G., 1996. *Improving locational speci_city of map data - a multiresolution, metadata-driven approach and notation*. International Journal of Geographic Information Systems Vol 10, pp.253-268.

Drolet Céline 2009. *Méthodes et outils de création, validation et étalonnage de populations synthétiques pour des géo-simulations multi-agents en milieu urbain.*

Ester M., Kriegel H.-P., Sander J., Xu X., 1998. *Clustering for Mining in Large Spatial Database.* Special Issue on Data Mining, KI-Journal, ScienTec Publishing, No. 1 (1998).

Ester M., Frommelt A., Kriegel H.-P., Sander J., 1998. *Algorithms for Characterization and Trend Detection in Spatial Databases*, Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, New York, NY, (1998)

Finkel, R. A., and Bentley, J. L.1974. *Quad trees: A data structure for retrieval on*

composite keys. Acta Inf. Vol 4(1), pp. 1-9.

Friedman, J. H., Bentley, J. L., and Finkel R.A., 1977. *An algorithm for finding best matches* in Sow, Vol 3(3), pp. 209-226.

Fuchs. H., Abram, G. D., and Grant, E. D., 1983. *Near Real-Time Shaded Display of Rigid Objects*, ACM Computer Graphics. Vol 17 (3), pp.65-72.

Fekete. G., 1990. Rendering and managing spherical data with sphere quadtrees. In Proceedings of Visualization, pp.176–186.

Goodchild and S. Yang, 1992. *A hierarchical spatial data structure for global geographic information systems*. CVGIP: Graphical Models and Image Processing, Vol 54(1), pp.31–44.

Gold C. 2006. *Voronoi hierarchies*. In Proceedings GIScience, Munster, Germany.

Gerstner. T, 2003. *Multi-resolution visualization and compression of global topographic data*. GeoInformatica Vol 7, pp. 7–32.

Gibson L. and Lucas D, 1982. *Vectorization of raster images using hierarchical methods*. Computer Graphics and Image Processing, Vol 20, pp, 9–82.

Han J.Koperski K., and Stefanovic N, 1997. *GeoMiner: A System Prototype for Spatial Data Mining*. Int'l Conf. on Management of Data (SIGMOD'97),Tucson, Arizona (1997) System prototype demonstration.

Hartman . P.N, Tanimoto S.L, 1984. *A Hexagonal Pyramid Data Structure for Image Processing,* IEEE trans. On Systems, Man, and Cybernetics, Vol14(2), 1984, pp, 247-256.

Johnson S. C. 1967. *Hierarchical Clustering Schemes*. Psychometrika, Vol 2, pp 241-254.

L. De Floriani and P. Magillo, 2002. *Multiresolution mesh representation: Models and data structures*. In M.Floater, A.Iske, and E.Qwak, editors, Tutorials on Multiresolution in Geometric Modelling,Springer-Verlag, 2002, pp 363- 418.

MacQueen. J. B. 1967. *Some Methods for classification and Analysis of Multivariate Observations,* Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability Vol 1, pp 281-297.

Moulin B Chaker,W., Perron J, Pelletier P., Hogan, J. et Gbei, E. 2003 *MAGS Project: Multi-Agent GeoSimulation and Crowd Simulation*, Spatial Cognition & Computation, Vol. 3, No. 4, pp. 359-375.

Matsuyama. T. Hao, L. V., and Nagao, M. 1984. *A File Organization for Geographic Information Systems Based on Spatial Proximity*', Computer Vision, Graphics and Image Processing, 26, pp.303-318.

Mark. D. M. and Abel, D. J., 1990. *A Comparative Analysis of Some Two-Dimensional Orderings,* International Journal of Geographical Information Systems, 4 (1), pp.21-31.

Otoo E and Zhu H. 1993. *Indexing on Spherical Surfaces Using Semi-Quadcodes,* In: Abel J. and Beng C. O.(Eds.), Advances in spatial Databases 3th International Symposium, SSD`93, Singapore, pp. 509-529.

Rhind D.W 1981. *Geographic Information Systems in Britain*, In Quantitative Geography: a British view, Wrigley, N. and R.J. Bennett (eds.) (London: Routledge and Kegan Paul), pp. 17-47.

Samet H, 1984. The *Quadtree and Related Hierarchical Data Structures ACM SIGMOD,* 10, pp 12-18.

Samet H and M. Lee 1988. *Navigating through Triangle Meshes Implemented as Linear Quadtrees* ACM SIGMOD, 10, pp11-18.

Samet H., 1990 *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

Strohmeier 1996. *Cycle de vie du logiciel.* In: A. Strohmeier, D. Buchs (Editors), Génie logiciel : principes, méthodes et techniques. Presses Polytechniques et Universitaires Romandes, Lausanne, pp. 1-28.

Schrack, G. 1992. *Finding neighbors of equal size in linear quadtrees and octrees in constant time,* CVGIP: Image Understanding Vol 55(3), pp.221-230.

Tsui Y , & Paul H. & Allan J.B 2002. *Adaptive Recursive Tessellations (ART) for Geographical Information Systems.*

Von Neumann J. 1996. *Theory of Self-Reproducing Automata.* Champaign-Urbana, University of Illinois Press.

Wolfram S. 1984. *Cellular automata as models of complexity*. Nature Vol 311, pp.419-425.

Annexes

Annexe A : Tables références et code de vieillissement de la population

Annexe A.1 Tables références

RoleMenage01	
NomRoleMenage	**CodeRole**
Personne seule	1
Conjoint travailleur	2
Conjoint étudiant	3
Conjoint au domicile	4
Conjoint retraité	5
Conjoint autre	6
Conjointe travailleuse	7
Conjointe étudiante	8
Conjointe au domicile	9
Conjointe retraitée	10
Conjointe autre	11
Enfant de moins de 6 ans	12
Enfant de 6 à 15 ans	13
Enfant de 16 à 20 ans aux études	14
Père monoparental	15
Mère monoparental	16
Adulte femme - travailleuse	17
Adulte homme - travailleur	18
Adulte femme - étudiante	19
Adulte homme - étudiant	20
Adulte femme - retraitée	21
Adulte homme - retraité	22
Adulte femme - sans occupation	23
Adulte homme - sans occupation	24

Tableau A.1.1 des rôles des ménages de la population synthétique

Profession	Catégorie	Code profession	Code Catégorie
Directeur	Cadre	A	1
Propriétaire	Cadre	B	1
Contremaître - surveillant	Contremaître	C	2
Éducation (professeur)	Professionnel	D	3
Santé (médecin, dentiste)	Professionnel	E	3
Droit (avocat, notaire)	Professionnel	F	3
Ingénieur	Professionnel	G	3
Autre professionnel	Professionnel	H	3
Technicien	Employé spécialisé	I	4
Comptable	Employé spécialisé	J	4
Secrétariat - bureau	Employé spécialisé	K	4
Infirmier	Employé spécialisé	L	4
Autre employé spécialisé	Employé spécialisé	M	4
Secrétariat - bureau	Employé non spécialisé	N	6
Restauration	Employé non spécialisé	O	6
Vente	Employé non spécialisé	P	6
Représentant	Employé non spécialisé	Q	6
Conducteur - chauffeur	Employé non spécialisé	R	6
Autre employé non spécialisé	Employé non spécialisé	S	6
Électricien, plombier, menuisier	Ouvrier qualifié	T	5
Mécanicien	Ouvrier qualifié	U	5
Autre ouvrier qualifié	Ouvrier qualifié	V	5
Manoeuvre, journalier	Ouvrier non qualifié	W	7
Autre ouvrier non qualifié	Ouvrier non qualifié	X	7
Militaire	Non classé	Y	0
Non classé	Non classé	Z	0

Tableau A.1.2 Code de classification d'emplois

Annexe A.2 Code source de vieillissement de la population

```
public bool Insert(String SQLR)
 {
      try
   {
       OleDbCommand cmd = connection.CreateCommand();
       cmd.CommandText = SQLR;
       dbReader = cmd.ExecuteReader();
       Object CellValue;
       while (dbReader.Read() )
       {
        Random rnd = new Random();
        CellValue = dbReader.GetValue(0);
        if (CellValue.ToString().Equals("3"))
        { int val = rnd.Next(10, 16);
        String rq = "insert into AgePersonnesJeunes01 (AgeP) values(" + val + ")";
        Insert(rq); }
        else if (CellValue.ToString().Equals("4")){
        int val = rnd.Next(15, 20);
        String rq = "insert into AgePersonnesJeunes01 (AgeP) values(" + val + ")";
        Insert(rq); }}
        fin = true ;
       }
    catch (Exception ex)
    { Console.WriteLine(ex.Message); }
     return fin;
 }
```

Figure A.2.1 Méthode qui assigne des valeurs aux cellules selon leurs types

```
public void Insert(String SQLR)
   {
                try
                        {
                OleDbCommand cmd = connection.CreateCommand();
                cmd.CommandText = SQLR;
               cmd.ExecuteNonQuery();
                 }
             catch (Exception ex)
            { Console.WriteLine(ex.Message); }
  }
```

Figure A.2.2 Methode pour executer une requête de type insertion ou mise à jour

Annexe B Dossier de conception

Annexe B.1 Diagramme de classes UML des patrons utilisés

Pattern Builder : Séparation entre représentation interne et instance

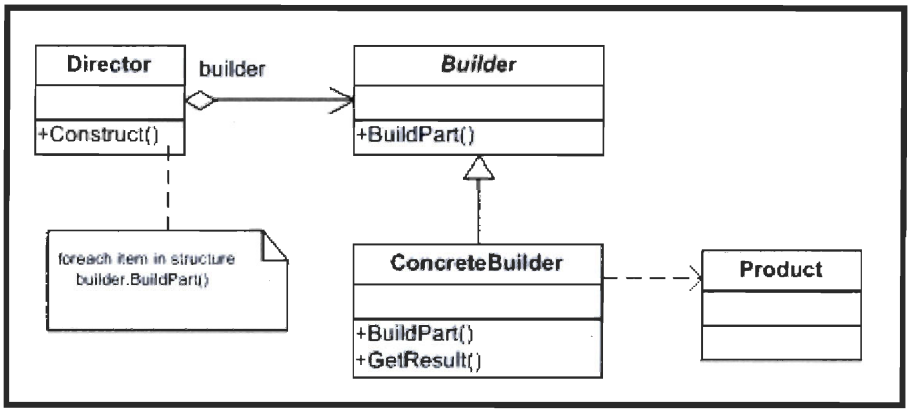


Figure B.1.1 Patron builder

Ce pattern est utilisé pour la création d'une variété d'objets complexes à partir d'un objet source. L'objet source peut consister en une variété de parties contribuant individuellement à la création de chaque objet complet grâce à un ensemble d'appels à l'interface commune de la classe abstraite Builder

Pattern Factory Methode : Délégation de la création aux sous-classes

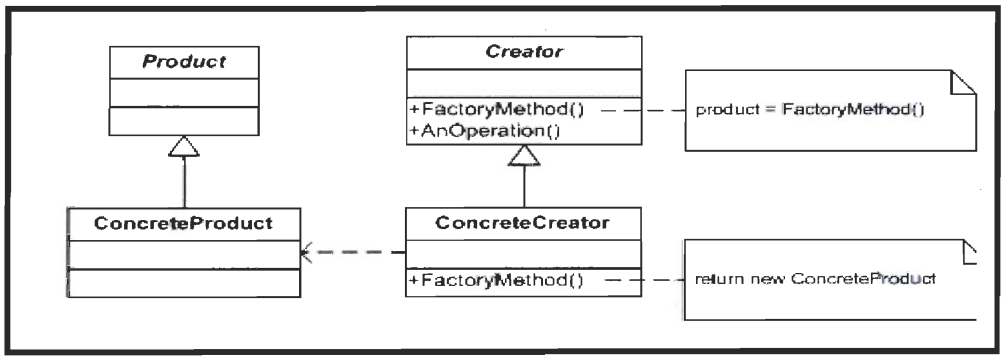


Figure B.1.2 patron factory

Il a pour rôle l'instanciation d'objets divers dont le type n'est pas prédéfini : les objets sont créés dynamiquement en fonction des paramètres passés à la fabrique.

Pattern Adapter : Adapter une interface existante à des nouveaux besoins

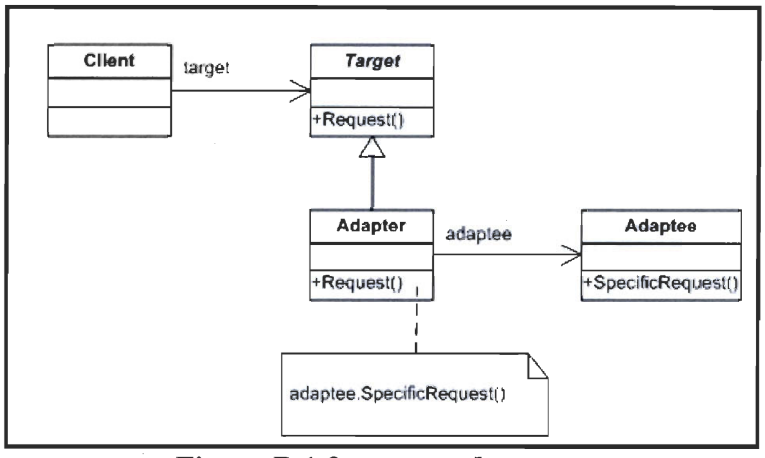


Figure B.1.3 patron adapter

Il permet de convertir l'interface d'une classe en une autre interface que le client attend. Il assure une compatibilité d'interfaces

Pattern Singleton : Assurer une instance unique

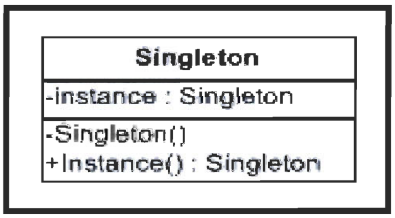


Figure B.1.4 pattern singleton

Il est utilisé pour restreindre l'instanciation d'une classe à un seul objet (ou bien quelques objets seulement). Il est utilisé lorsque l'on a besoin d'exactement un objet pour coordonner des opérations dans un système. Le modèle est parfois utilisé pour son efficacité, lorsque le système est plus rapide ou occupe moins de mémoire avec peu d'objets qu'avec beaucoup d'objets similaires.

Annexe B.2 Dictionnaire des classes

SingeltonList : Liste contenant tous les triangles issus d'un partitionnement de l'espace.

Triangles : Classe qui représente une cellule triangulaire.

TComparables : Interfaces pour comparer des triangles.

LTriangularQuadtree : Classe qui définit un quadtree linéaire.

Index : Classe qui crée un index en Z de Peano et en Column Order

HexagonalCel : Classe qui définit une cellule hexagonale

HexagonalCollection : Classe qui définit une grille hexagonale à un niveau donné.

GridLevelFactory : Une factory qui crée une grille hexagonale selon le niveau.

HierachicalGrid : Un client à la Factory qui enregistre l'ensemble des grilles.

BDController : Classe qui gère les opérations d'accès à la Base de données.

DModel : Classe responsable de la persistance des données.

Cluster : Classe responsable du regroupement

Semantic : Classe responsable de charger les données sémantiques dans les grilles

SingeltonListConfiguration : Liste globale des options de l'usager

FrameModler : Classe qui représente l'interface d'accès aux données

Fviwer : Classe qui affiche les grilles.

Forme Principale : Fenêtre qui contient les autres fenêtres.

Annexe C : Guide de l'utilisateur

I Introduction

Dans le cadre du projet *MUSCAMGS*, l'application que nous avons développée permet de créer des grilles hexagonales hiérarchiques. Elle permet également d'informer l'environnement spatial avec des données sémantiques. Par ailleurs, cette application localise sur la carte de la ville de Québec des zones qui ont des propriétés sémantiques communes. Dans le cadre de ce projet, nous avons utilisé la technologie.NET et le langage de programmation C#. Dans ce qui suit, nous allons présenter les étapes d'utilisation de notre application. En effet, nous allons présenter les interfaces qui montreront comment un usager pourrait utiliser notre application.

II Étapes d'utilisation de l'application :

1 Lancement de l'application :

Au lancement de l'application, un écran d'accueil apparaît. Il contient deux boutons, un pour lancer l'application et un autre pour la quitter.

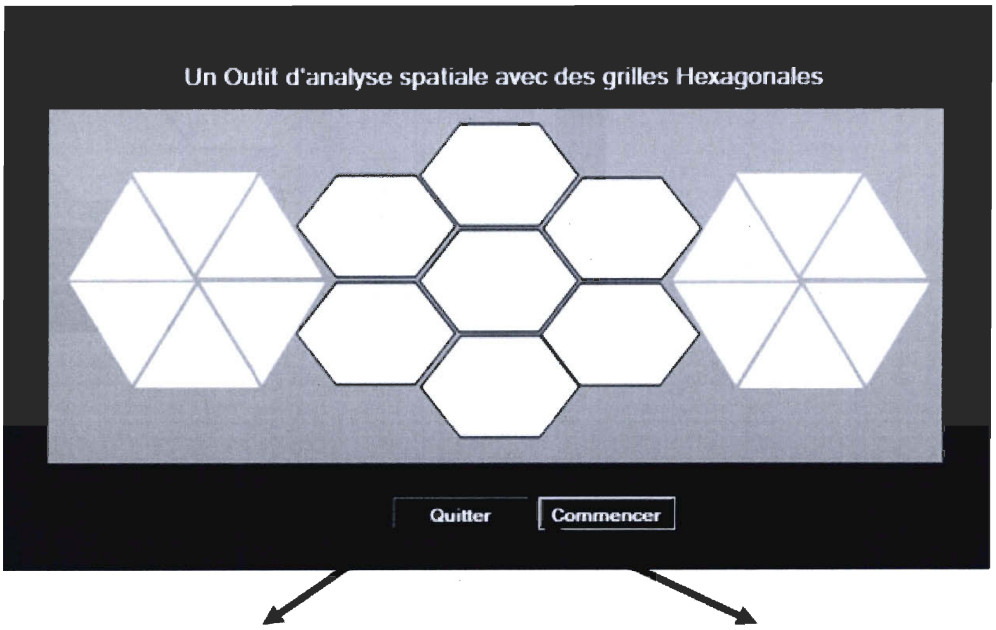


Figure C.1 écran de lancement de l'application

2 Écran principal

Un écran principal s'affiche à la suite de l'opération en 2.1. Il permet de contenir les opérations nécessaires à l'utilisation de cet outil par l'entremise de sa barre d'outil.

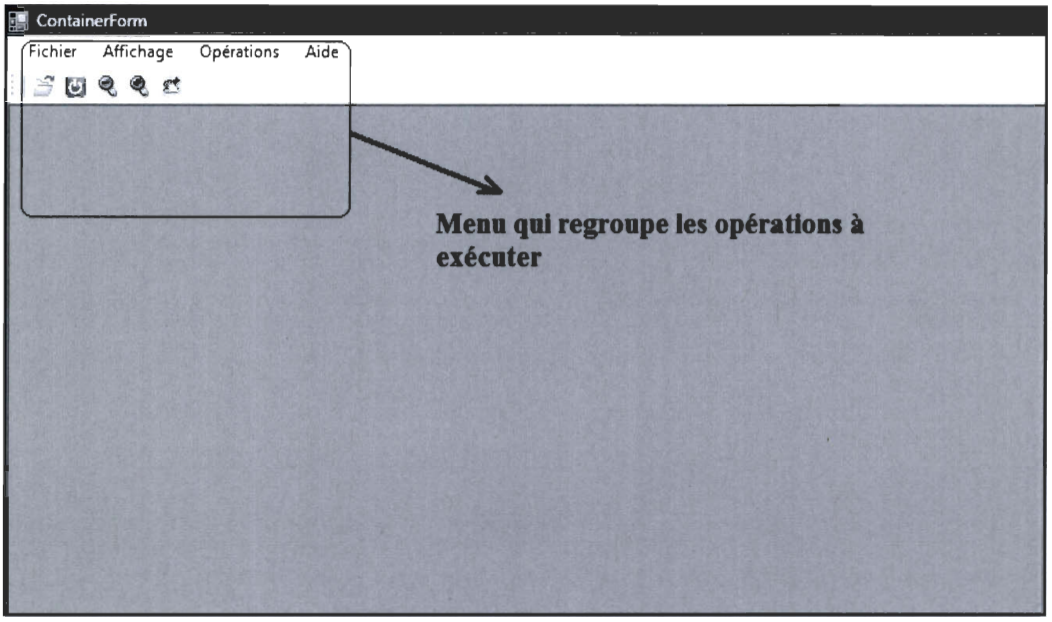


Figure C.2 écran principal.

3 Opérations à exécuter :

3.1 Menu Fichier : Ce menu permet trois opérations. Il permet d'ouvrir un fichier existant, capturer un écran et quitter l'application.

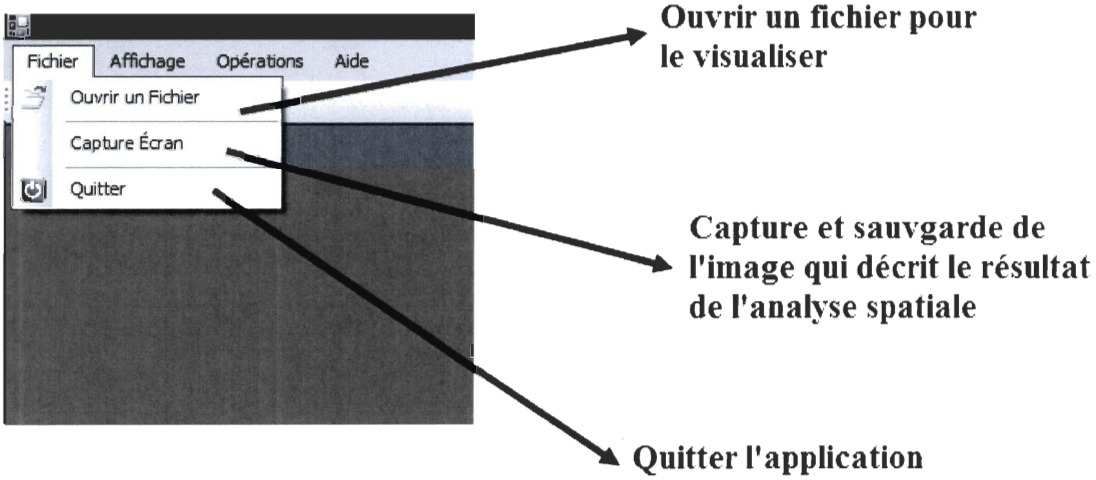


Figure C.3 écran de choix des opérations

3.2 Menu Affichage: Il permet de regrouper les opérations de zoom (avant et arrière) sur la fenêtre principale qui contient les résultats de notre analyse. Il permet également de contenir l'opération responsable du Pan.

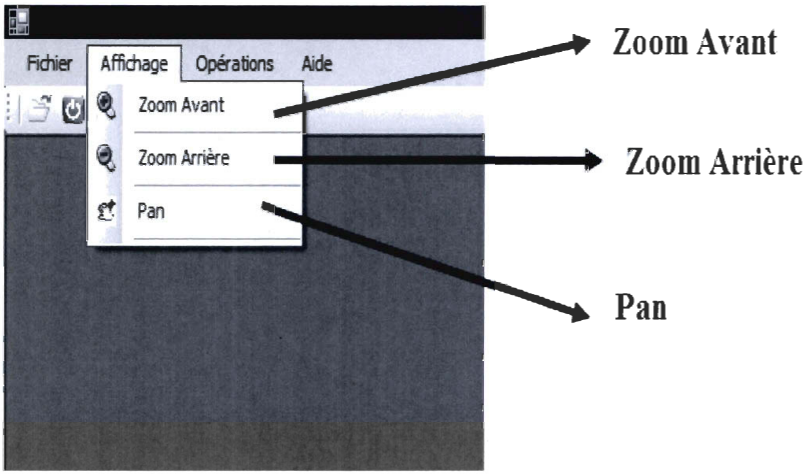


Figure C.4 écran de zoom

3.3 Menu Analyse : Ce menu est le menu le plus important de notre application. Il contient les opérations qui permettent de créer des grilles hexagonales, visualiser et analyser les données. Nous présentons d'abord les opérations de ce menu, ensuite nous allons présenter les écrans de chaque opération séparément.

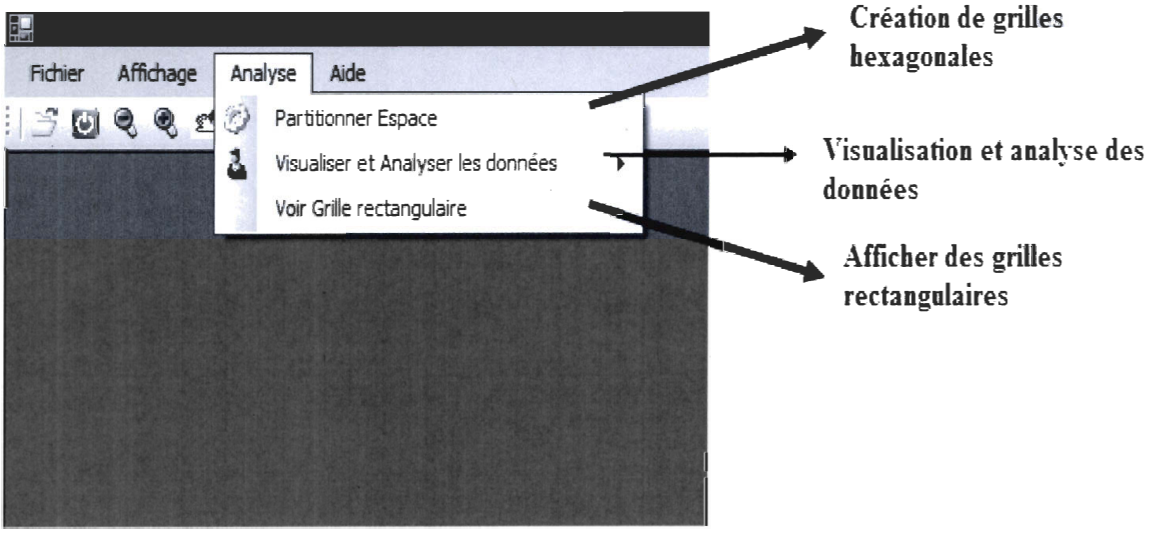


Figure C.5 écran de choix des actions pour l'usager

3.3.1 Opération partitionner l'espace :

Cette opération permet de créer des grilles hexagonales selon les algorithmes que nous avons présentés dans le chapitre 4. Quand l'usager clique sur *Partitionner espace* l'écran suivant apparaît :

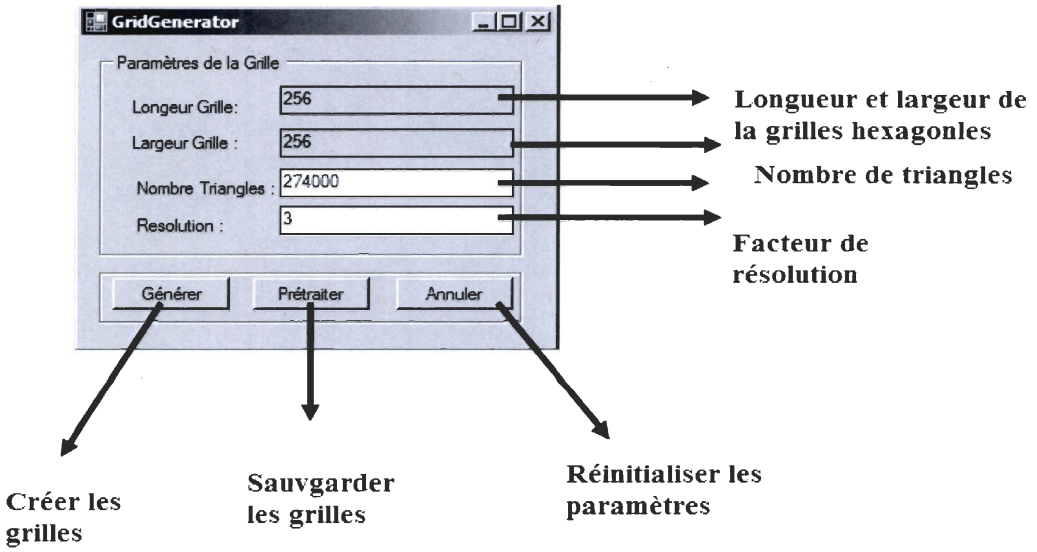


Figure C.6 écran de configuration de la création des grilles

3.3.2 Opération visualiser et analyser les données :

L'utilisateur clique sur le menu *Analyse* ensuite *visualiser et analyser les données* et finalement *Analyser les données*. L'écran suivant montre ce processus.

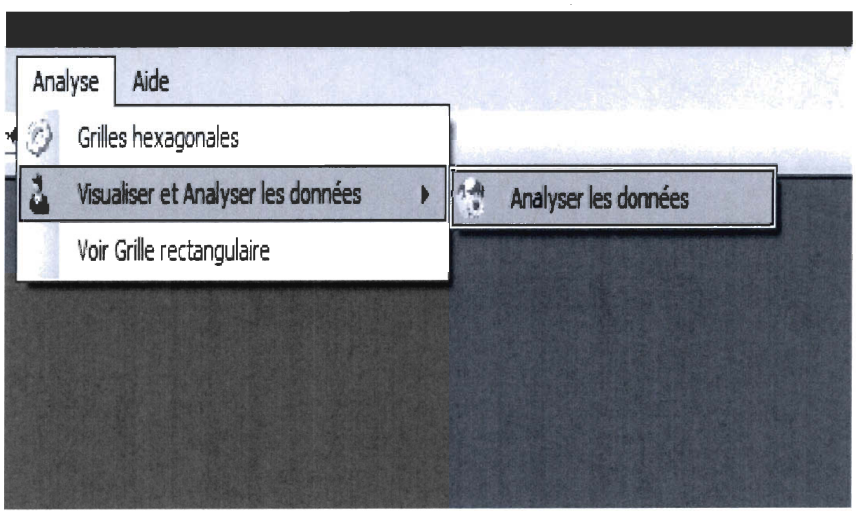


Figure C.7 écran d'analyse des données

À la suite de cette action l'écran *Configurations* apparaît (Écran H). Cette action est responsable de configurer l'analyse des données selon divers paramètres spatiaux et sémantiques.

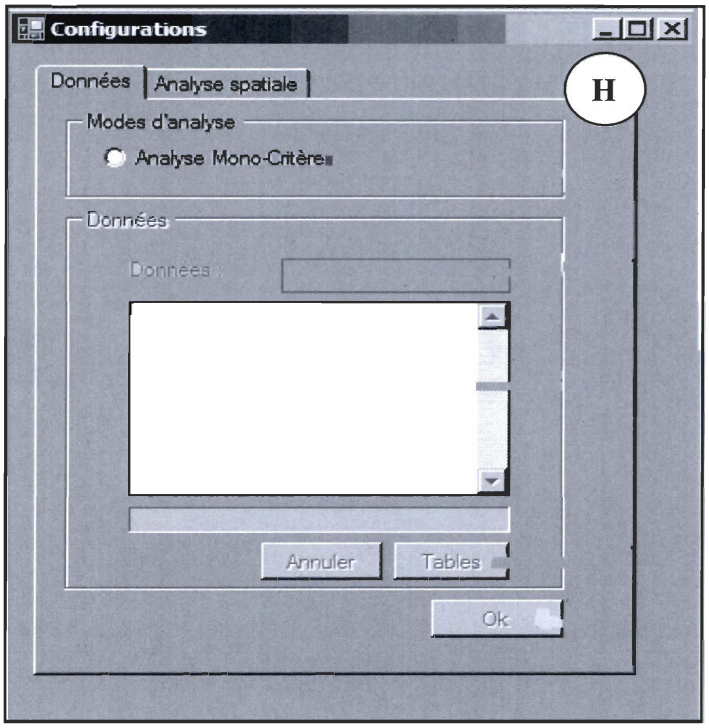


Figure C.8 écran de saisie des variables à analyser

Ceci est illustré par l'interface suivante. Elle est formée par deux autres interfaces. L'un concerne les paramètres sémantiques et l'autre est utilisée pour les paramètres spatiaux.

a) Interface pour les paramètres sémantiques

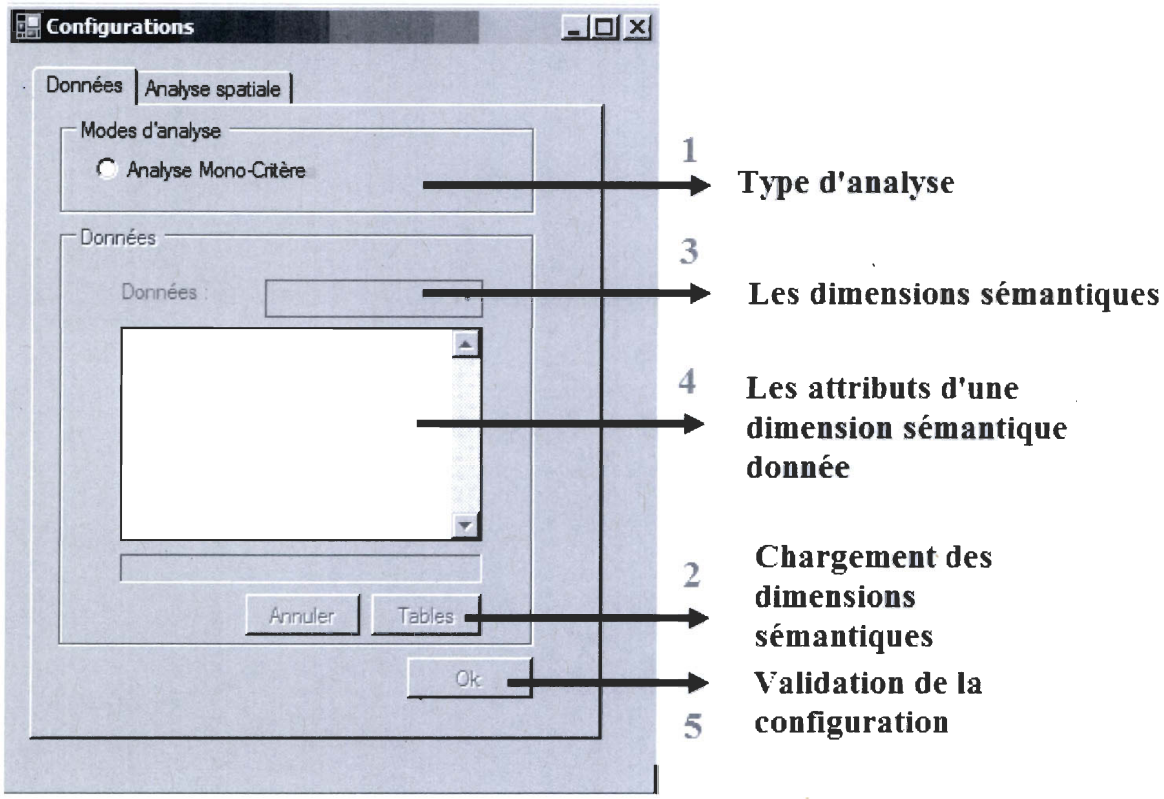


Figure C.9 Interface des variables sémantiques

Les chiffres de 1 à 5 montrent l'enchaînement des actions suivantes de l'usager.
1 : Type d'analyse. Il s'agit du type d'analyse à effectuer. Pour le moment le système supporte une analyse monocritère et non multicritère.
2 : Le bouton *Table* permet de charger les dimensions sémantiques du jeu de données dans la case données. C'est à l'aide de ces dimensions que l'usager va choisir une seule dimension pour l'analyser.
3 : Ce menu permet à l'usager de choisir une seule dimension sémantique.
4 : L'usager choisit un attribut d'une dimension sémantique.
5 : l'usager valide sa configuration.
Dans ce qui suit, pour chaque action identifiée par un chiffre nous allons montrer l'interface obtenue et en même temps un scénario d'utilisation de cette interface.
Après avoir cliqué sur le type d'analyse (action 1), activer le chargement des dimensions sémantiques, et le choix d'une seule dimension (action 2 et 3), l'interface **A** s'affiche.

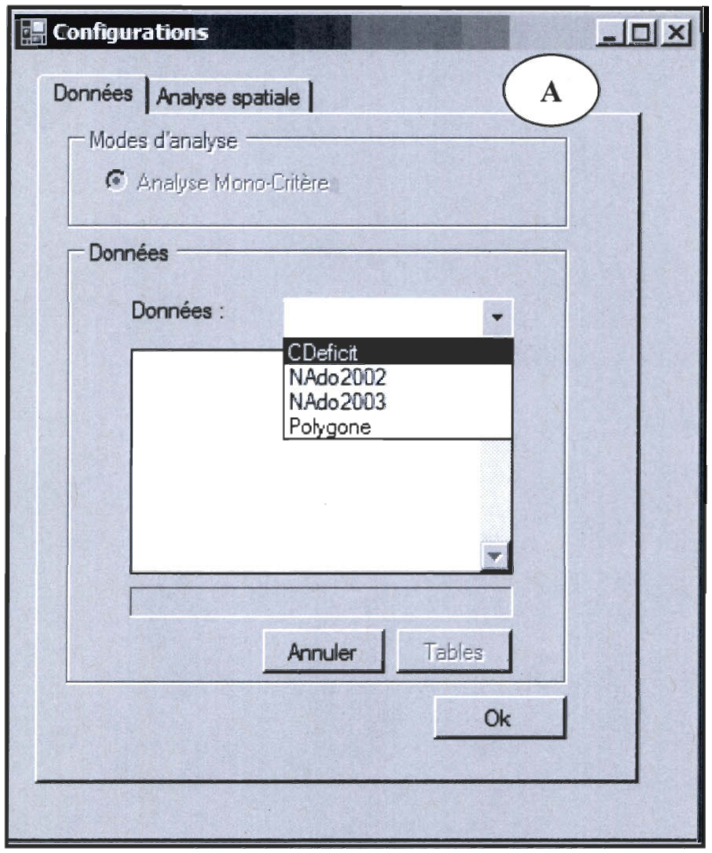


Figure C.410 écran de configuration des données à analyser

L'utilisateur choisie le champ *CDEFICIT* qui représente les catégories de déficit de voitures. L'écran B s'affiche et l'usager peut effectuer l'action 4. Finalement, l'usager peut confirmer son choix en cliquant sur le bouton *OK* en bas à droite

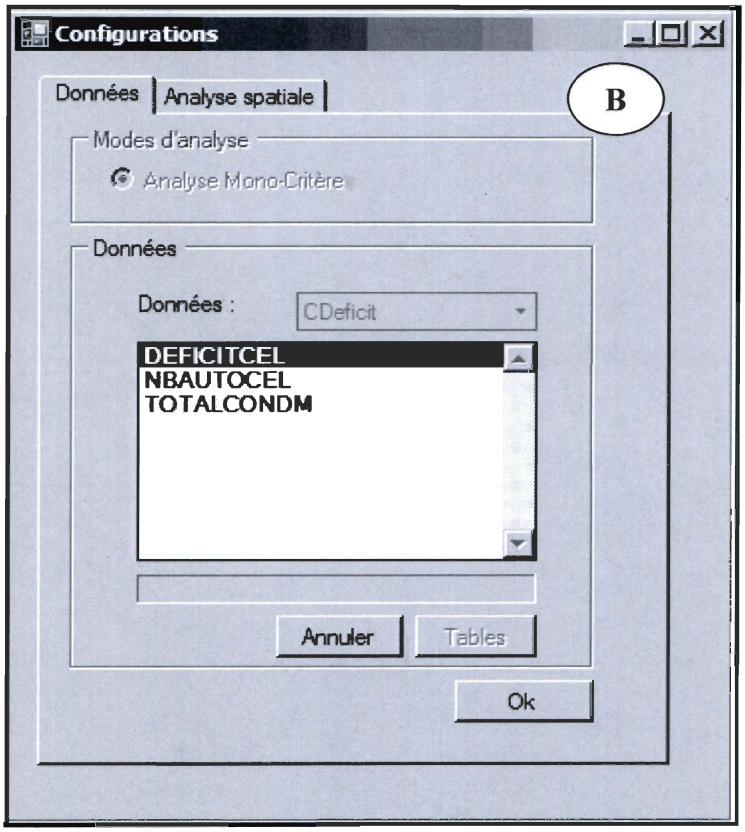


Figure C.11 écran de validation des données à analyser

b) Interface pour les paramètres spatiaux :

L'écran présenté par la figure suivante permet de choisir les paramètres spatiaux comme les couleurs, le type de regroupement et les distances.

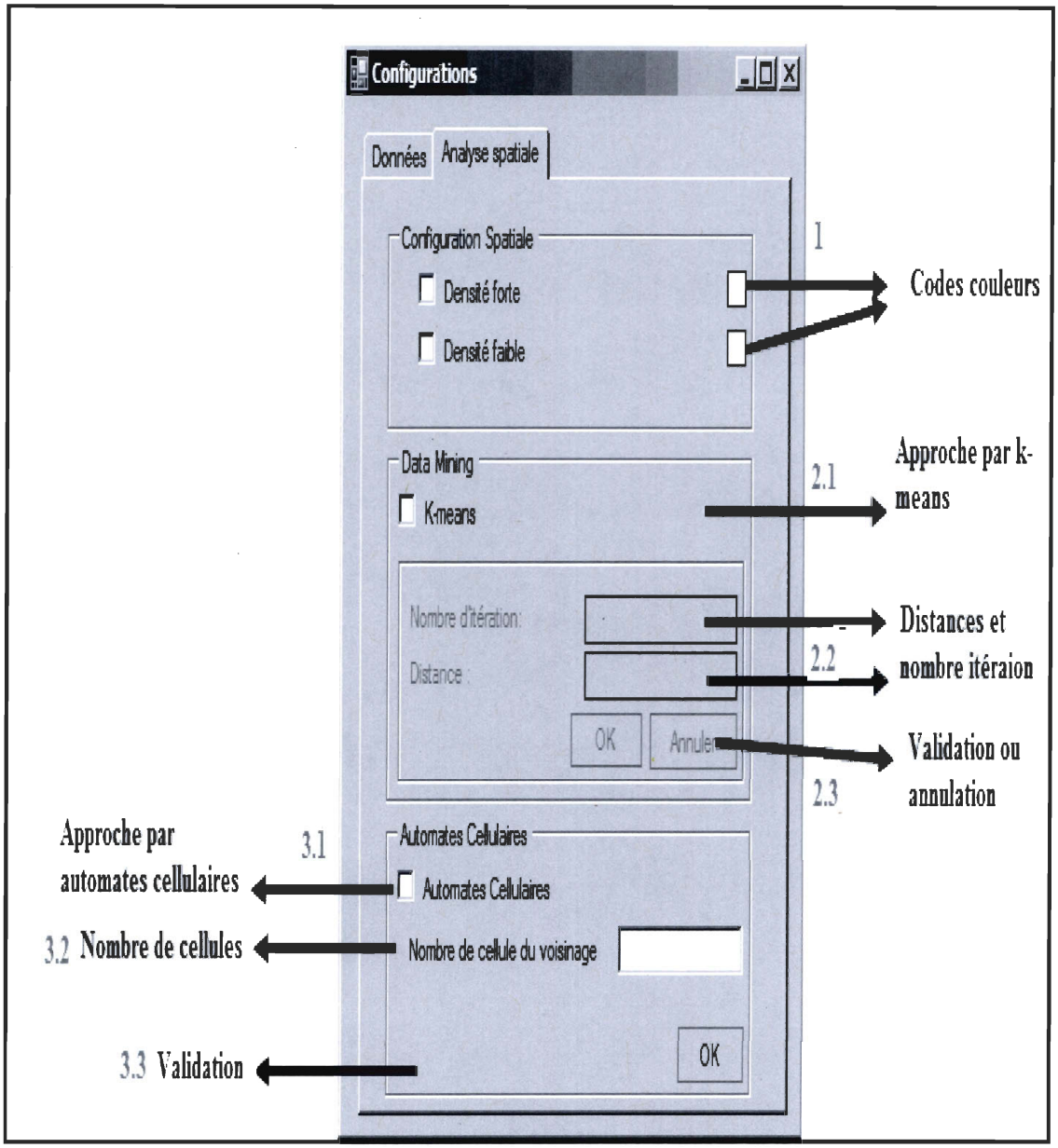


Figure C.12 écran de configuration des paramètres spatiaux

Cet écran permet à l'usager de choisir un mode de regroupement. Si l'utilisateur exécute les opérations 1, 2.1, 2.2 et 2.3 il obtient l'interface C et D. L'interface C permet à l'usager de choisir des codes couleurs.

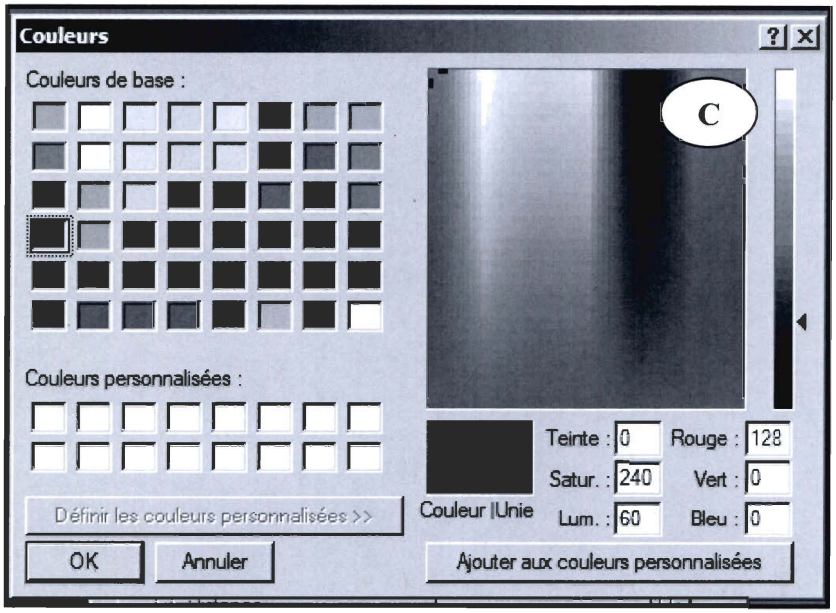


Figure C.13 écran de choix des couleurs

Quant à l'interface D, elle permet à l'usager de fixer les distances pour le regroupement. À la suite de ces opérations, l'interface E s'affiche. Elle permet de visualiser le résultat du regroupement.

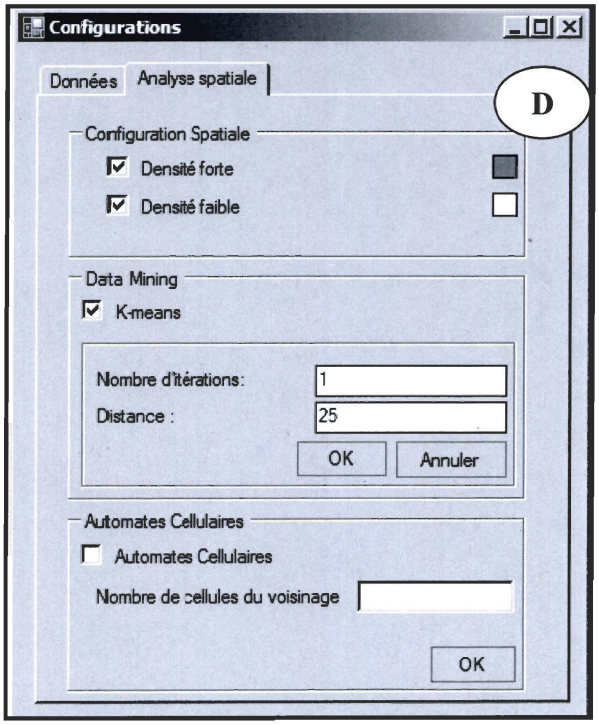


Figure C.14 écran de saisie des distances

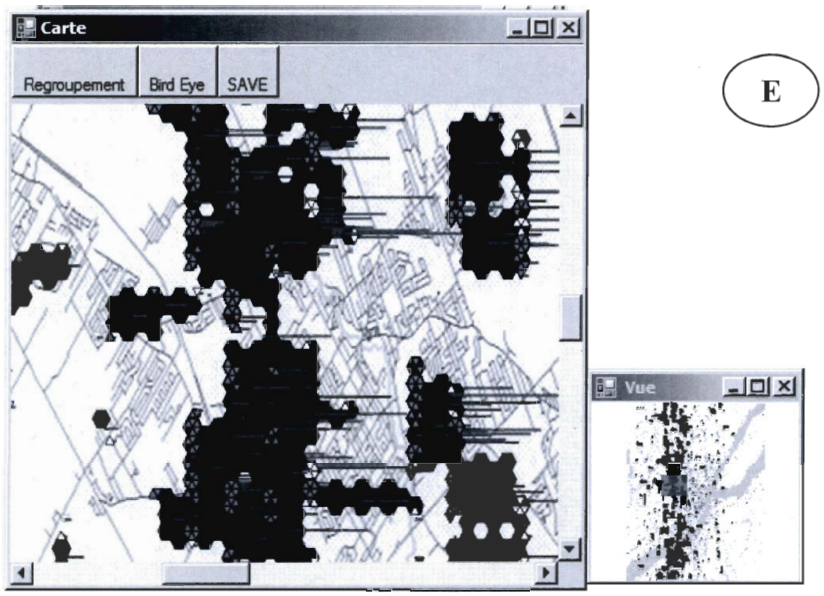


Figure C.15 écran de visualisation

Quand l'usager exécute les opérations 1, 3.1, 3.2 et 3.3 l'interface F s'affiche. Elle correspond au résultat de regroupement selon l'approche par automates cellulaires.

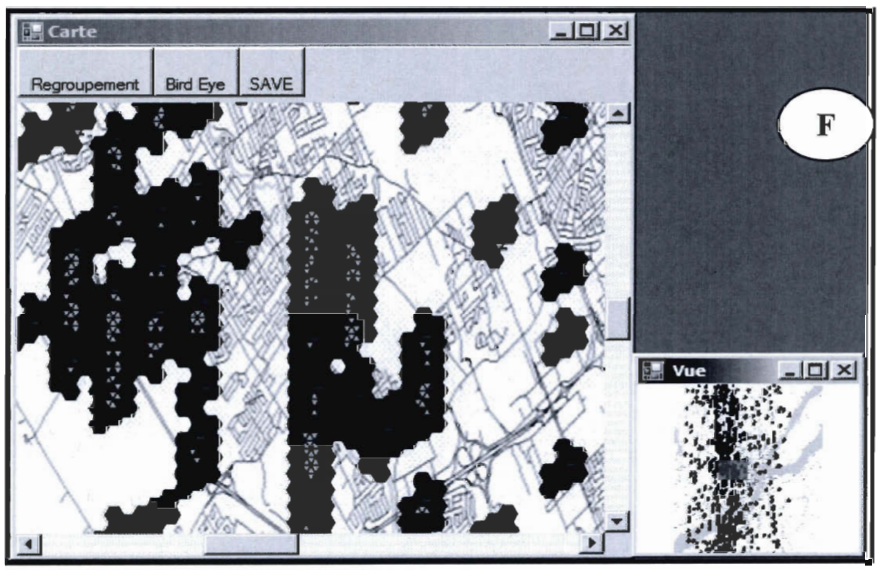


Figure C.16 écran de visualisation des automates cellulaires

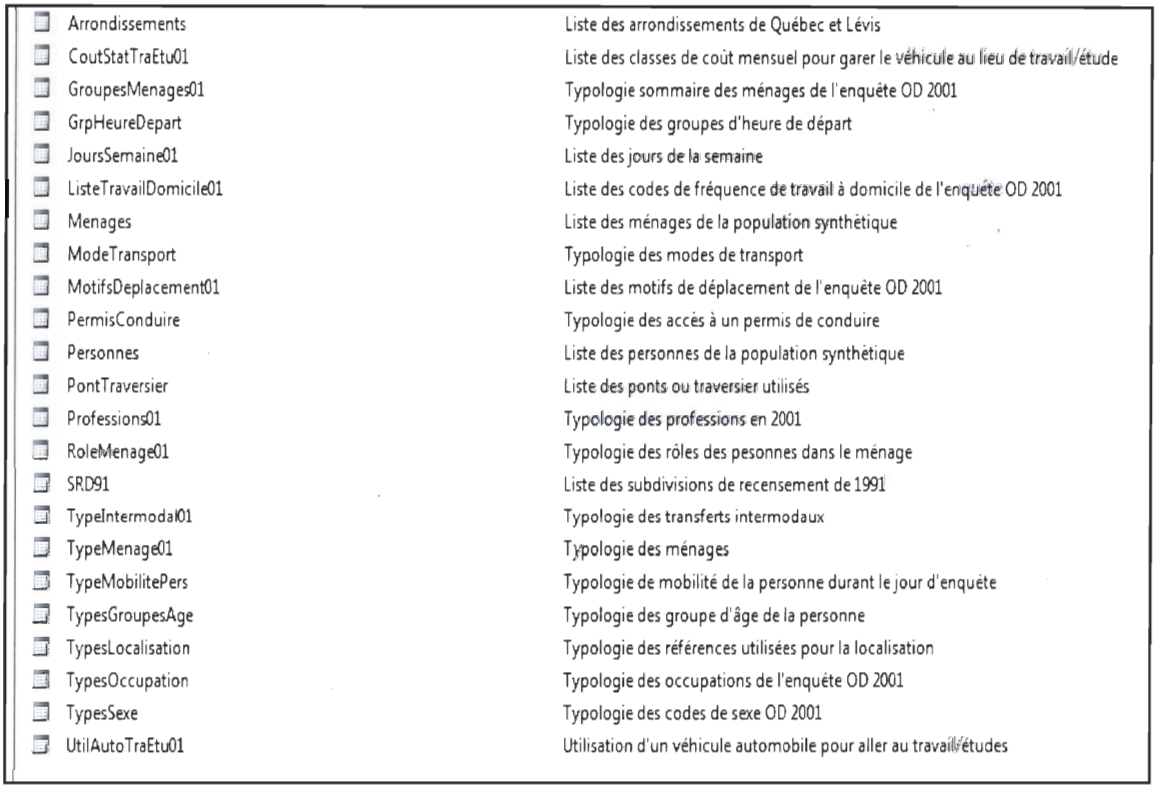
Annexe D Organisation des données et requêtes SQL pour le vieillissement de la population

Requête	Table résultat
SELECT ResidGrHex, count(UniqueMena) AS MenagesAdos FROM CellulesAdos02 WHERE ageP = 16 GROUP BY (ResidGrHex) HAVING count(*)>1;	NbAdos02
SELECT ResidGrHex, count(UniqueMena) AS MenagesAdos FROM CellulesAdos03 WHERE ageP = 16 GROUP BY (ResidGrHex) HAVING count(*)>1;	NbAdos03
SELECT ResidGrHex, count(UniqueMena) AS MenagesAdos FROM CellulesAdos04 WHERE ageP = 16 GROUP BY (ResidGrHex) HAVING count(*)>1;	NbAdos04
SELECT ResidGrHex, count(UniqueMena) AS MenagesAdos FROM CellulesAdos05 WHERE ageP = 16 GROUP BY (ResidGrHex) HAVING count(*)>1;	NbAdos05
SELECT ResidGrHex, count(UniqueMena) AS MenagesAdos FROM CellulesAdos06 WHERE ageP = 16 GROUP BY (ResidGrHex) HAVING count(*)>1;	NbAdos06

Tableau D.1 Requêtes pour l'obtention du nombre des adolescents entre 2001 et 2006

Requête	Table résultat
SELECT ResidGrHex, Cellulesados.UniqueMena, AgePersonnesJeunes02.UniquePers, ageP FROM AgePersonnesJeunes02, Cellulesados WHERE AgePersonnesJeunes02.UniquePers = CellulesAdos.UniquePers;	CellulesAdos02
SELECT ResidGrHex, Cellulesados.UniqueMena, AgePersonnesJeunes03.UniquePers, ageP FROM AgePersonnesJeunes03, Cellulesados WHERE AgePersonnesJeunes03.UniquePers = CellulesAdos.UniquePers;	CellulesAdos03
SELECT ResidGrHex, Cellulesados.UniqueMena, AgePersonnesJeunes04.UniquePers, ageP FROM AgePersonnesJeunes04, Cellulesados WHERE AgePersonnesJeunes04.UniquePers = CellulesAdos.UniquePers;	CellulesAdos04
SELECT ResidGrHex, Cellulesados.UniqueMena, AgePersonnesJeunes05.UniquePers, ageP FROM AgePersonnesJeunes05, Cellulesados WHERE AgePersonnesJeunes05.UniquePers = CellulesAdos.UniquePers;	CellulesAdos05
SELECT ResidGrHex, Cellulesados.UniqueMena, AgePersonnesJeunes06.UniquePers, ageP FROM AgePersonnesJeunes06, Cellulesados WHERE AgePersonnesJeunes06.UniquePers = CellulesAdos.UniquePers;	CellulesAdos06

Tableau D.2 Ménages d'intérêts entre 2001 et 2006



Arrondissements	Liste des arrondissements de Québec et Lévis
CoutStatTraEtu01	Liste des classes de coût mensuel pour garer le véhicule au lieu de travail/étude
GroupesMenages01	Typologie sommaire des ménages de l'enquête OD 2001
GrpHeureDepart	Typologie des groupes d'heure de départ
JoursSemaine01	Liste des jours de la semaine
ListeTravailDomicile01	Liste des codes de fréquence de travail à domicile de l'enquête OD 2001
Menages	Liste des ménages de la population synthétique
ModeTransport	Typologie des modes de transport
MotifsDeplacement01	Liste des motifs de déplacement de l'enquête OD 2001
PermisConduire	Typologie des accès à un permis de conduire
Personnes	Liste des personnes de la population synthétique
PontTraversier	Liste des ponts ou traversier utilisés
Professions01	Typologie des professions en 2001
RoleMenage01	Typologie des rôles des pesonnes dans le ménage
SRD91	Liste des subdivisions de recensement de 1991
TypeIntermodal01	Typologie des transferts intermodaux
TypeMenage01	Typologie des ménages
TypeMobilitePers	Typologie de mobilité de la personne durant le jour d'enquête
TypesGroupesAge	Typologie des groupe d'âge de la personne
TypesLocalisation	Typologie des références utilisées pour la localisation
TypesOccupation	Typologie des occupations de l'enquête OD 2001
TypesSexe	Typologie des codes de sexe OD 2001
UtilAutoTraEtu01	Utilisation d'un véhicule automobile pour aller au travail/études

Figure D.1 : Organisation des données thématiques dans la base de données