Apprentissage discriminant sur des graphes pour la caractérisation des paysages propices à l'étalement urbain

Daniel Vantroys

Stage de Master recherche

Rapport de stage

Apprentissage discriminant sur des graphes pour la caractérisation des paysages propices à l'étalement urbain

Auteur :
Daniel Vantroys

Superviseurs :
Thomas Guyet
Véronique Masson
René Quiniou
DREAM

Résumé

Un paysage peut-être décrit comme une organisation spatiale d'éléments paysagers. Pour caractériser des paysages propices à l'étalement urbain, des outils liés à leur représentation sous forme de structures apparentées à des graphes ou à un ensemble de relations doivent être considérés. Différentes approches ont été étudiées : pour les graphes, une caractérisation à partir de la fouille de motifs pour l'utilisation d'un classifieur classique, des SVM avec noyaux adaptés ou des arbres de décision particuliers, et la programmation logique inductive pour la représentation sous forme de relations. L'approche relationnelle a été retenue pour son expressivité et les outils puissants disponibles (Prolog). Une implémentation de Foil (First-Order Inductive Learner) avec quelques extensions est devenue l'objectif du stage.

Table des matières

Remerciements

Je tiens à remercier Thomas Guyet pour son encadrement, notamment pour son aide concernant l'implémentation du travail.

Je voudrais également remercier Véronique Masson et René Quiniou pour leur encadrement et leurs explications sur certains détails théoriques.

Et merci à toute l'équipe Dream pour leur accueil !

1 Problématique

On peut décrire un paysage par les éléments qui le compose (tels que des parcelles agricoles, des routes, des bâtiments, des forêts...) et les relations qui les lient (distance, adjacence...). Le but du stage est d'identifier des profils caractéristiques de paysages propices à l'étalement urbain, i.e. la progression spatiale des surfaces artificialisées (routes, bâtiments...), pour expliquer l'urbanisation d'une région.

Pour cela, des cartes ont été obtenues par des méthodes de télédétection en 1992, 2000 et 2010 pour Lorient. La figure 1 correspond à une image de Lorient en 1992 avec analyse de l'occupation du sol (les surfaces en rouge sont celles identifiées comme étant du bâti). Une analyse diachronique des images identifie les parcelles cadastrales de 2000 comme positives ou négatives selon que l'on ait observé ou non un étalement urbain entre 2000 et 2010. On cherche à apprendre à partir de ces cartes comment classifier n'importe quel nouvel exemple : sur une carte a priori inconnue, y aura-t-il étalement urbain dans les années à venir ?

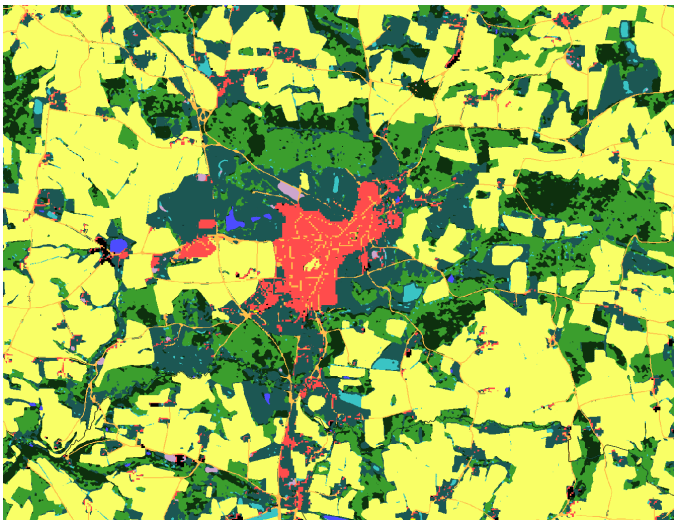


FIGURE 1 – Image de Lorient en 1992 obtenue par télédétection. Les surfaces urbanisées sont en rouge

Nous avons donc un problème d'apprentissage supervisé. De nombreuses techniques existent pour les données qui se présentent sous forme de liste d'attributs, mais les cartes ne sont pas réductibles à de telles structures de données : il y aurait une trop grande perte d'information à utiliser un nombre fixe d'attributs pour décrire une carte.

Un premier moyen de représenter des données spatiales est d'utiliser des graphes étiquetés de manière à laisser en évidence la caractéristique spatiale (par exemple, un nœud peut être étiqueté par son type, c'est-à-dire

s'il représente une route, un bâtiment, etc., et une arête par une précision sémantique de la relation qui lie deux objets spatiaux, comme l'adjacence). Cette structure de graphe peut être étendue en incluant un certain nombre de relations observables, voire en utilisant un modèle purement relationnel, très expressif, utilisant des relations implicites (par exemple, en composant des relations).

Le but est non seulement d'obtenir un classifieur qui prédit si une zone sera urbanisée ou non, mais aussi que l'on puisse comprendre la décision dudit classifieur, que l'on puisse l'interpréter pour expliquer l'étalement urbain. Ainsi, ce qu'on appellera l'interprétabilité du modèle doit être un critère pour le choix de l'approche retenue.

2 Approches

La problématique suggère la recherche d'une solution donnant des résultats interprétables facilement. Les solutions étudiées dans la littérature en amont du stage sont donc mentionnées dans cette section selon le pouvoir d'interprétabilité dont elles sont dotées. Les graphes étant une forme plus simple que la représentation logique, on mentionne donc en premier les travaux sur les graphes, dont les techniques de fouille, l'utilisation de méthodes à noyau pour graphes et des modèles d'arbres de décision adaptés aux graphes, avant d'étudier le cas de la programmation logique inductive.

2.1 Méthodes de fouilles de motifs pour caractériser les graphes

La fouille de données consiste à chercher des caractéristiques fréquentes dans le jeu de données, ces caractéristiques pouvant être utilisées a posteriori pour discriminer les données. Si les données sont sous forme de graphes, on peut, par exemple, chercher les sous-graphes fréquents.

Dans le domaine de la fouille de graphes, il existe quelques travaux qui se sont intéressés à la prise en compte d'une information spatiale.

Apriori-based Graph Mining (AGM) [19] est une adaptation de l'algorithme Apriori pour la fouille de graphes. Apriori est un algorithme de fouille de données qui extrait des règles d'associations fréquentes. AGM permet d'extraire des sous-graphes fréquents et des règles d'associations entre ceux-ci (par exemple, la présence d'un motif implique la présence d'un autre). Il s'est révélé efficace (relire l'article pour plus de précision) sur une étude de composés moléculaires. (reciter l'article ? ou ne le citer qu'à la fin de ce paragraphe ?)

La géométrie étant une information de nature spatiale, on peut citer les travaux portant sur les graphes géométriques (les nœuds ont des coordonnées, donc des configurations géométriques sont présentes) tels que ceux de M. Kuramochi et G. Karypis [20], et H.Arimura, T. Uno et S. Shimozono [4].

M. Kuramochi et G. Karypis ont développé un algorithme, gFSG, également inspiré d'Apriori qui recherche les sous-graphes géométriques fréquents, autorisant les transformations simples (translations, rotations, changements d'échelle) et en admettant une tolérance sur la mesure de similarité. Il peut être utilisé avec de très grandes bases de données de graphes représentant notamment des données spatiales.

H.Arimura, T. Uno et S. Shimozono ont conçu un algorithme ne devant retenir que les sous-graphes géométriques de taille maximale, gFSG présentant pour eux le risque de trouver trop de sous-graphes.

J. Huan et al. [18] ont une approche plus orientée théorie de l'information. Ils définissent une information mutuelle entre deux graphes qui leur permet d'introduire la notion de sous-graphe k-cohérent : tout sous-graphe de taille k dont l'information mutuelle avec le graphe d'origine est supérieur à un seuil strictement positif. Le but devient alors d'extraire tous les sous-graphes k-cohérents. À l'aide de SVM (*Support Vector Machines*, cf. 2.2) pour classifier des familles de protéines, leur approche donne de bons résultats.

2.2 Méthodes à noyaux spécifiques aux graphes

Les SVM (*Support Vector Machines*, machines à vecteurs de support ou séparateurs à vastes marges) consistent à séparer des données (pourvues d'attributs numériques) en deux classes selon un hyperplan, en maximisant la distance de cet hyperplan aux exemples et en autorisant quelques erreurs. Souvent, les données ne sont pas linéairement séparables, donc on construit une fonction $\phi(.)$ qui projette les données dans un espace de dimension plus grande, appelé l'espace de redescription, dans lequel la séparation linéaire pourra avoir lieu. La recherche de l'hyperplan séparateur nécessite des calculs de produits scalaires. Or, dans l'espace de redescription, à cause de la plus grande dimension (potentiellement infinie), les calculs sont alourdis. Mais il existe des fonctions $K(.,.)$ bilinéaires symétriques semi-définies positives telles que leur application à deux éléments de l'espace d'origine valent le produit scalaire de leurs correspondants dans l'espace de redescription. $K(.,.)$ est une *fonction noyau*. Formellement :

$$\forall x, y \; K(x,y) = \langle \phi(x) | \phi(y) \rangle$$

L'utilisation de noyaux permet donc de séparer des vecteurs dans l'espace de redescription sans avoir à les représenter explicitement dans cet espace. Une fonction noyau peut suffire pour séparer en deux classes des données, peu importe la représentation initiale des données, et en particulier si ces données sont des graphes.

Pour des raisons d'efficacité du calcul des fonctions noyaux, les noyaux de graphes décrivent les graphes au travers de sous-structures présentes dans le graphe. La distance induite par le noyau entre deux graphes est donc une approximation.

K. M. Borgwardt et al. [8] exposent des noyaux adaptés aux graphes utilisant, par exemple, des marches aléatoires (suites aléatoires de sommets reliés par des arêtes).

$$k_{graph}(G_1, G_2) = \sum_{walk_1 \in G_1} \sum_{walk_2 \in G_2} k_{walk}(walk_1, walk_2)$$

où $k_{walk}(walk_1, walk_2)$ vaut le produit des fonctions noyaux sur chaque couple de sommets. L'étude de K. M. Borgwardt et al. portait sur des protéines (sous forme de graphes) donc le noyau utilisé se trouve modifié en une combinaison de plusieurs noyaux portant sur les attributs spécifiques aux graphes (types d'arêtes, de nœuds). Un autre point de vue, valable surtout dès que les graphes deviennent grands, est d'étudier la distribution de "petits" motifs au sein des graphes. C'est l'idée des noyaux sur les *graphlets* (cite), sous-graphes de k sommets ($k \in \{3, 4, 5\}$). L'énumération de ces sous-graphes étant un problème complexe, N. Shervashidze et al. [28] procèdent à un échantillonnage des sous-graphes et appliquent dessus leur fonction noyau.

Cette technique se révèle souvent efficace, mais ne tient pas compte d'éventuelles spécificités spatiales des données (pas d'intervention dans les calculs d'étiquettes sur les arêtes) et les résultats produits pour séparer les classes ne sont pas interprétables.

2.3 Arbres de décision

Les arbres de décision sont des classifieurs discriminants facilement interprétables par l'être humain : le principe est de faire suivre une série de tests, hiérarchiquement ordonnés par un arbre, à l'objet considéré en empruntant le chemin (partant de la racine) correspondant aux résultats de ces tests pour finalement arriver à une feuille qui donne sa classe (ici, un test correspond à la présence d'un sous-graphe).

L'apprentissage d'arbre de décision consiste à diviser récursivement, et le plus efficacement possible, les exemples de l'ensemble d'apprentissage par des tests jusqu'à n'avoir plus que des sous-ensembles d'exemples tous (ou presque) de la même classe.

Dans toutes les méthodes, trois opérateurs sont majeurs :

- décider si un nœud est terminal (*i.e.* s'il peut être étiqueté comme une feuille de l'arbre).
- si un nœud n'est pas terminal, lui associer un test
- si un nœud est terminal, lui associer une classe

La sélection du test associé à un nœud est une étape essentielle. Dans le cas général des données structurées sous forme de liste d'attributs, cela correspond à trouver un attribut de partage. On utilise alors une fonction d'évaluation qui favorise les attributs discriminants. Cette fonction mesure le gain

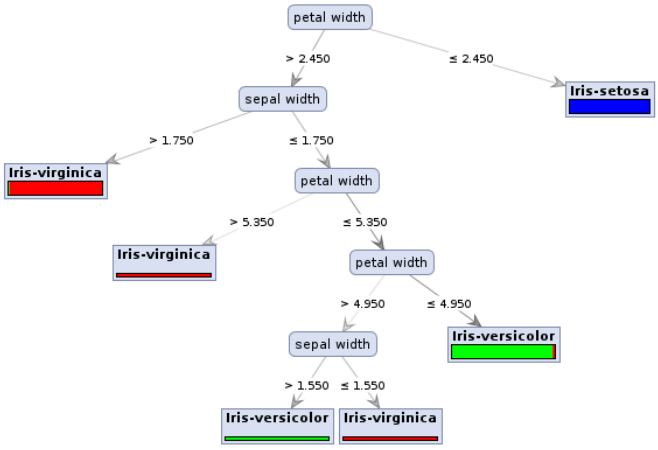


FIGURE 2 – Exemple d'arbre de décision classique déterminant le type de fleur en fonction des dimensions d'un pétale

en information des attributs en utilisant en général des critères statistiques comme l'entropie.

GBI (Graph-Based Induction) est une technique d'apprentissage efficace qui extrait des motifs (sous-graphes) fréquents sur des graphes de manière gloutonne. Il est possible de construire en suivant cette approche des arbres de décision pour graphes ([14], [25]).

GBI et ses améliorations (B-GBI [1], Cl-GBI [2]) servent à trouver, de manière itérative, des motifs fréquents dans des graphes dont les nœuds sont étiquetés. Cl-GBI permet que ces motifs fréquents puissent se recouvrir. Notons que Cl-GBI est aussi bien adapté aux graphes orientés que non orientés, et certaines relations spatiales, comme l'adjacence, ne sont pas orientées, alors que d'autres, comme l'inclusion, le sont. Un arbre de décision discriminant selon les motifs présents dans les graphes se construit récursivement en utilisant ce principe. On calcule par Cl-GBI les motifs fréquents des graphes à disposition, on détermine quel motif est le plus discriminant, on sépare l'ensemble de graphes en deux selon que ce motif est présent ou non, on calcule récursivement l'arbre associé à chacune de ces parties, on obtient les sous-arbres correspondant aux fils du nœud courant (motif présent/absent).

La recherche de motifs est une boucle sur ces trois étapes pour N_e itérations (paramètre à fixer selon le temps de calcul, la taille des motifs acceptés...) :

1. Extraire toutes les paires de nœuds reliés pas une arête dans les graphes en comptant les occurrences. À partir de la deuxième itération, au

1. Beam-wise Graph-Based Induction
2. Chunkingless Graph-Based Induction

moins l'un des nœuds de ces paires doit être un nouveau pseudo nœud créé au point 2 de l'itération précédente.

2. Considérer les b paires les plus fréquentes parmi celles qui viennent d'être extraites et, à partir de la deuxième itération, des paires extraites aux itérations précédentes mais non sélectionnées à cette étape. Enregistrer le motif correspondant comme un nouveau pseudo nœud.
3. Donner une étiquette à chaque nouveau pseudo nœud.

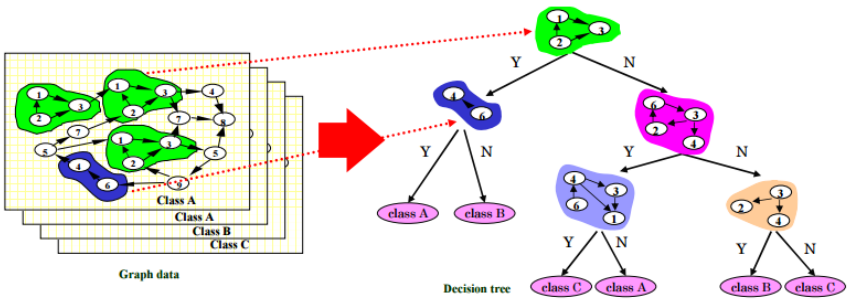


FIGURE 3 – Arbre de décision sur des motifs fréquents [25]

Pour utiliser cet arbre de décision sur un nouveau graphe, pour chaque motif successif rencontré sur l'arbre, on procède comme suit : on utilise une version modifiée du processus de recherche (on se restreint aux sous-motifs du motif discriminant) sur le graphe pour vérifier s'il contient ou non le motif, avec des paramètre b et N_e suffisamment élevés pour avoir peu de chance de manquer quelque chose.

Dans le cas de graphes géographiques, où les nœuds sont étiquetés selon ce qu'ils représentent, et où les arêtes sont également potentiellement étiquetées selon la signification de la liaison, Cl-GBI peut être utilisé avec les avantages et les inconvénients qui vont avec le fait que ce soit un algorithme glouton et que le temps de calcul est paramétrable.

2.4 Programmation Logique Inductive

La PLI [24] est une technique d'apprentissage supervisé qui permet d'induire des hypothèses, ou règles, à partir d'exemples. Ces exemples sont étiquetés positifs ou négatifs selon qu'ils représentent ou non des observations du concept à inférer. La PLI permet en particulier de dépasser certaines limites des formalismes de représentation des connaissances dans les systèmes d'apprentissage en utilisant la logique du premier ordre. Cette modélisation permet de rendre compte plus naturellement des relations entre objets. La PLI met en œuvre l'induction en logique des prédicats qui est utilisée comme langage de représentation des exemples (faits Prolog) et des hypothèses (clauses de Horn). L'inférence de nouvelles règles correspond à une

recherche de clauses dans un espace organisé selon une relation de généralité. Les algorithmes de recherche descendante, comme FOIL, partent d'une clause générale vers des clauses plus spécifiques en utilisant des opérations comme l'ajout de littéraux à la clause de départ ou l'application de substitution pour transformer des variables en constantes ou pour unifier plusieurs variables. Les clauses ainsi générées sont ensuite testées sur les exemples de façon à généraliser le maximum d'exemples positifs et peu ou pas d'exemples négatifs.

En PLI, la taille de l'espace de recherche rend nécessaire l'utilisation d'heuristiques afin de guider le parcours de l'espace au cours de la recherche. Un type d'heuristique très utilisé concerne les fonctions d'évaluation, qui mesure en quelque sorte l'utilité de chaque clause examinée. L'utilité d'une hypothèse peut s'exprimer par la quantité d'information apportée, par la compression de la base de connaissance réalisée ou par son pouvoir de discrimination comme l'heuristique Gain de FOIL (*First-Order Inductive Learner*) [26].

L'algorithme 1 décrit le principe de l'inférence de règles par FOIL [26] [31].

Algorithme 1: Algorithme FOIL

Entrées : Ensemble d'exemples positifs E_+ et négatifs E_-
Résultat : Ensemble S de règles

```
S ← ∅
Pos ← E+
tant que Pos ≠ ∅ faire
    Neg ← E−
    r ← nouvelle règle initialisée
    tant que Neg ≠ ∅ faire
        "Trouver" le "meilleur" littéral à ajouter à r
        Retirer de Neg les exemples non couverts
    fin
    Ajouter r à S
    Retirer de Pos les exemples couverts
fin
```

Le littéral à ajouter présente un intérêt soit car il introduit une nouvelle variable (un nouvel attribut), soit car il permet de bien discriminer les exemples négatifs des exemples positifs. La formalisation de "bien discriminer" se fait par la définition d'une fonction de gain prenant en paramètres le nombre d'exemples couverts selon que le littéral soit appliqué ou non, que ces exemples soient positifs ou non. Une définition du gain est explicitée à la partie 3.2.1.

L'utilisation de la PLI pour les données spatiales permet une vue plus claire que la structure de graphe multi-étiqueté. La représentation relationnelle est plus expressive, elle permet par exemple d'exploiter des relations

implicites, différents niveaux de granularité dans les relations [22].

Code 1 – Exemple de code Prolog

```
1  zone(z0).
2  centre(z0,5,6).
3  parcel(p0).
4  centre(p0,2,2).
5  inzone(z0,p0).
6  parcel(p1).
7  centre(p1,4,4).
8  inzone(z0,p1).
9  adjacent(p0,p1).
10 adjacent(p1,p0).
11 parcel(p2).
12 centre(p2,7,7).
13 inzone(z0,p2).
14 adjacent(p1,p2).
15 adjacent(p2,p1).
16 locbetween(p1,p0,p2).
17 tree(t0).
18 centre(t0,10,4).
19
20 distance(A,B,D) :- centre(A,Xa,Ya), centre(B,Xb,Yb),
       X2 is Xa-Xb, Y2 is Ya-Yb, D is sqrt(X2*X2+Y2*Y2).
```

L'exemple Prolog présente la déclaration d'une zone z_0, de parcelles, p_0, p_1 et p_2 ; incluses dans cette zone (relation *inzone*), quelques relations entre ces parcelles (adjacence de p_0 et p_1, adjacence de p_1 et p_2, le fait que p_1 soit située entre p_0 et p_2), d'un arbre t_0, la définition d'une règle exprimant la distance entre deux entités spatiales (distance entre deux centres, comme le centre de deux parcelles, le centre d'une parcelle et le centre d'une zone...). Des notions de Prolog sont présentées en annexe A. La figure 4 illustre une représentation de cet exemple sous une forme graphique. Seule l'information d'adjacence peut être visualisée et la relation ternaire (*locbetween*) ne peut être ainsi représentée. Les coordonnées des centres seraient des attributs des nœuds et la relation de distance se traduirait par des arêtes entre toutes les paires de nœuds étiquetées par la valeur de la distance.

La PLI a été utilisé dans des problématiques de classification supervisée de données à caractère spatial : N. Chelghoum et al. [9] ont implémenté S-TILDE, inspiré de TILDE (*Top-down Induction Logical DEcision tree*) [6], une méthode de classification par arbre de décision basée sur la logique du premier ordre. S-TILDE a été utilisé pour une classification supervisée d'emplacements de coquillages contaminés (ou non) dans la lagune de Thau. R. Trépos [31] a travaillé sur différentes techniques de PLI pour un système d'aide à la gestion des activités agricoles et des aménagements sur un bassin versant.

R. Frank, M. Ester et A. Knobbe ont utilisé la PLI dans le cadre de clas-

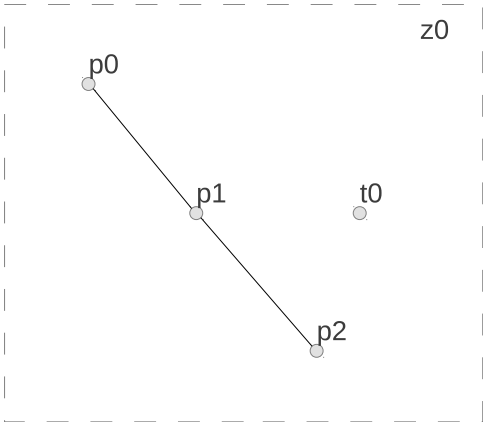


FIGURE 4 – Graphe équivalent au Prolog. Un arc indique la relation d'adjacence

sification spatiale pour déterminer des règles sur les boutiques, centres commerciaux, etc. qui ont été victimes de vol [13]. Leur algorithme, UnMASC (*Unified Multifeature Aggregation based Spatial Classifier*), repose sur FOIL, méthode générale d'inférence de règles. UnMASC inclue des agrégats, *i.e.* des méta-prédicats permettant d'exploiter une fonction sur les mêmes attributs d'un ensemble d'objets, par exemple, exprimer la superficie moyenne des parcelles d'une zone).

L'introduction d'agrégats permet d'avoir des règles qui font appels à de nombreux attributs d'un même exemple de manière synthétique, ce qui confère une plus grande généralité des règles : si `boutique(B), collier(B, C), prix(B, P), P>500` signifie que le prix d'un collier de la boutique considérée est supérieur à 500$, `boutique(B), min(P, collier(B, C), prix(B, P), MP), MP>500` signifie que le prix de tous les colliers de la boutique est supérieur à 500$ (car le prix minimum l'est).

Pour plus d'efficacité, UnMASC parallélise l'évaluation des gains des littéraux supplémentaires. Évaluer une règle consiste à déterminer son gain (cf. 3.2.1). Les règles peuvent être évaluées en parallèle car ces calculs sont indépendants, ce qui réduit le temps d'exécution. Les expériences de R. Frank, M. Ester et A. Knobbe montrent que l'implémentation parallèle sur une machine à huit processeurs est cinq fois plus rapides que l'implémentation série de l'algorithme.

Bien que le point de vue initial sur les données spatiales soit celui d'une structuration en graphes, les approches basée sur la PLI, qui étendent facilement la sémantique des différents composants des graphes (nœuds, arêtes) à l'aide de de moult relations, sont plus convaincantes au niveau de l'expres-

sivité et de l'interprétabilité : vérifier une règle est généralement plus simple que retrouver un sous-graphe.

3 Matériel et méthode

L'approche retenue pour le stage est une adaptation de UnMASC [13] à notre problématique. Deux critères, l'expressivité de la méthode (surtout avec l'introduction d'agrégats) et son interprétabilité, ont conduit à ce choix. Le processus de parallélisation peut aussi permettre d'alléger les temps de calcul.

Dans la suite de cette section, on détaillera la génération de la base d'exemples à partir des données utilisées, des "cartes" de Lorient de 2000 et 2010 traitées et annotées, puis on expliquera l'algorithme d'inférence de règles.

3.1 Construction d'un jeu d'apprentissage

Afin de répondre au problème, il est nécessaire d'obtenir des données et de les structurer de manière à pouvoir les traiter. Cette section explique l'origine de nos cartes (images de télédétection) et la construction d'un jeu d'exemples de classification.

3.1.1 Sources et format d'origine

Les données sont issues d'images obtenues par télédétection de la ville de Lorient en 2000 et 2010. Ces données ont été acquises et traitées par le laboratoire PSN [3] dans le cadre du projet PayTal (`www.paytal.fr`).

Ces images sont analysées pour décomposer les différentes occupations du sol. Onze catégories sont distinguées : sol artificiel, voie de communication, eau, forêt de feuillus, forêt de conifères, zones agricoles, végétation semi-naturelle/prairie, sol nu, kaolin/sol nu clair, plage, estran. La figure 5(a) illustre le résultat de l'analyse de la décomposition du sol sur une carte de Lorient en 2000 (les bâtiments, en rouge, sont concentrés au centre de l'image).

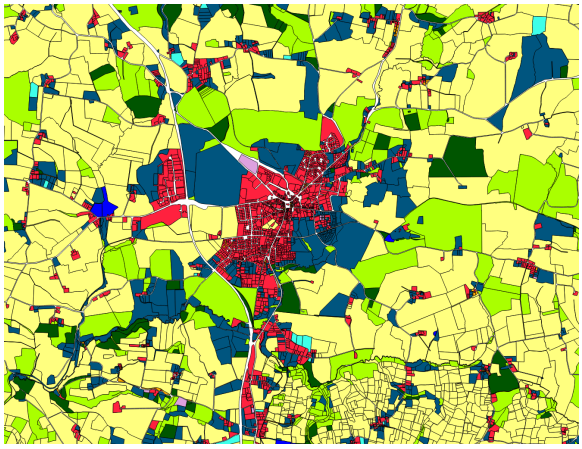
Les images sont ensuite vectorisées à l'aide du cadastre de 2010. À chaque parcelle ainsi délimitée est attribuée son occupation du sol selon l'occupation majoritaire du sol dans la parcelle cadastrale considérée. La figure 5(b) montre le résultat de ce découpage par rapport à la figure 5(a). Le procédé donne aux parcelles un degré d'"urbanisation" compris entre -1 et 1 portant sur la différence de surface urbanisée entre 2000 et 2010 :

$$d_{urb} = \frac{n_{pixels\ urbanisés\ en\ 2010} - n_{pixels\ urbanisés\ en\ 2000}}{n_{pixels\ urbanisés\ en\ 2010} + n_{pixels\ urbanisés\ en\ 2000}}$$

3. Physique et Spatialisation Numérique, UMR SAS, Agrocampus/INRA, Rennes

Ce degré donne la classe de la parcelle : une parcelle de 2000 est classée positivement (*i.e.* considérée comme urbanisée en 2010) si $d_{urb} > 0$. La figure 5(c) illustre l'attribution des classes sur des parcelles de Lorient. Le degré est normalisé pour être inchangé selon l'échelle considérée (car on considère des nombres de pixels). La normalisation choisie évite d'avoir des valeurs considérées comme nulles car trop petites (par exemple si la normalisation était faite par rapport à l'aire de la parcelle).

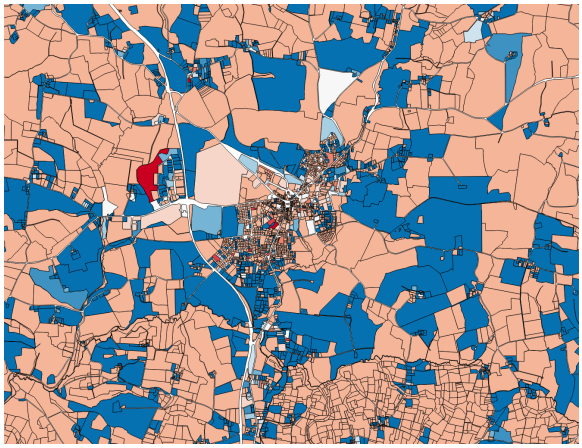
(a) Image obtenue par télédétection en 2000. Les surfaces en rouge correspondent à du bâti.



(b) Vectorisation de l'image précédente à partir du cadastre de 2010. Les types d'occupations du sol sont uniformisés pour chaque parcelle.

FIGURE 5 – Exemple du traitement d'un échantillon de la ville de Lorient

L'idée de la formalisation de ce degré est de pouvoir donner les classifications suivantes si les parcelles n'avaient qu'un état binaire urbaine/non



(c) Degré d'urbanisation entre 2000 et 2010 (bleu = positif, rouge = négatif, les couleurs pâles sont proches de 0, les plus foncées sont proches de 1 ou -1)

FIGURE 5 – Exemple du traitement d'un échantillon de la ville de Lorient (suite)

urbaine :

- urbaine en 2000 et urbaine en 2010 $\Rightarrow$ classe $\ominus$
- urbaine en 2000 et non urbaine en 2010 $\Rightarrow$ classe $\ominus$
- non urbaine en 2000 et urbaine en 2010 $\Rightarrow$ classe $\oplus$
- non urbaine en 2000 et non urbaine en 2010 $\Rightarrow$ classe $\ominus$

Les données sont également enrichies avec des informations qui ont pu être ajoutées automatiquement à l'aide d'autres données librement disponibles (dont la base de données parcellaire de l'IGN), telles que la spécification de certaines zones en écoles, en parcs naturels ou en zones commerciales.

Le cadastre numérisé de 2000 n'a pas pu être récupéré, d'où l'utilisation du cadastre de 2010 pour délimiter des parcelles sur les images de 2000. Même si le cadastre n'a pas subi des modifications drastiques en dix ans, cette utilisation peut engendrer quelques aberrations, comme la présence d'un lotissement récent qui transforme un champ en 2000 en un ensemble de petites parcelles.

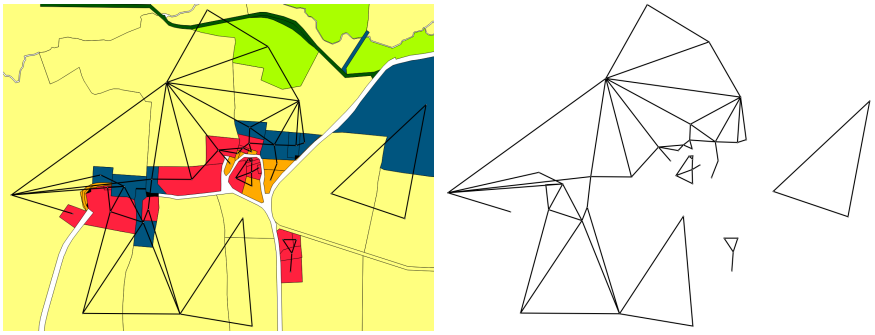
3.1.2 Construction d'un graphe

À partir des opérations précédentes, un grand graphe est construit : les nœuds représentent les parcelles et sont munis des attributs déjà calculés (classe d'occupation du sol, urbanisation positive/négative, attributs statuant si la parcelle est un parc naturel, une zone commerciale ou une école), ainsi que des attributs ajoutés tels que le périmètre et l'aire de la parcelle. Les arêtes désignent les relations entre les parcelles : adjacence, distance

entre les barycentres des parcelles...).

Cette étape limite l'approche à l'utilisation de relations binaires.

La figure 6 montre un exemple de construction de graphe. Les nœuds sont situés aux centroïdes des parcelles associées. Seule la relation d'adjacence est exprimée par les arêtes. Bien que ce ne soit pas représenté, les nœuds sont étiquetés notamment avec le type d'occupation du sol de la parcelle (couleur de la parcelle).



(a) Graphe sur image vectorielle (deux parcelles séparées par une route ne sont pas considérées comme adjacentes)

(b) Graphe

FIGURE 6 – Exemple de construction de graphe

3.1.3 Construction de la base d'exemples

Le graphe construit précédemment ne peut pas être utilisé directement pour la classification. Pour l'apprentissage supervisé, il est nécessaire de construire un ensemble de zones dont certaines représentent des exemples positifs et d'autres des exemples négatifs. Deux questions se posent alors : comment segmenter le zones et attribuer une classe (positive ou négative) aux zones obtenues.

Plusieurs types d'extraction de zones ont été envisagés :

- faire un pavage de l'espace, et définir une zone comme l'ensemble des parcelles inclues dans un élément du pavage.
- sélectionner une parcelle et considérer toutes les parcelles situées dans un certain rayon du centre de la parcelle sélectionnée.
- sélectionner un point et considérer toutes les parcelles situées dans un certain rayon de ce point.

Dans tous les cas, on obtient pour chaque zone un sous-graphe dont les nœuds sont les nœuds associés aux parcelles que la zone contient et les arêtes sont toutes les arêtes entre les différentes paires de nœuds du sous-graphe qui sont définies dans le graphe principal.

Différentes possibilités ont été évoquées pour l'attribution des classes aux zones ainsi choisies :

- la classe de la majorité des parcelles (avec éventuellement pondération par les aires)
- la classe de la parcelle centrale (celle qui contient le centre de la zone)

Dans nos expérimentations, nous avons choisi de définir les sous-graphes exemples par une parcelle centrale et de leur attribuer la classe de cette parcelle centrale.

L'ensemble de ces sous-graphes est ensuite transcrit automatiquement sous forme logique dans un programme Prolog (cf. annexe A pour quelques rappels de notations Prolog) de la manière suivante :

- un sous-graphe est représenté par le prédicat `instance` sur son identifiant. Par exemple, le fait `instance(g1)` est enregistré.
- un nœud d'un sous-graphe est introduit par le prédicat `inzone`. Par exemple, le fait `inzone(g1,g1_n1)` est enregistré.
- pour les sous-graphes pour lesquels l'urbanisation future est positive (la classe d'un sous-graphe est attribuée selon la classe de toutes les parcelles contenues dans le sous-graphe), le prédicat `positif` est utilisé. Par exemple, si `g1` désigne une instance positive, le fait `positif(g1)` est enregistré. Pour une instance négative, rien n'est enregistré, l'exécution du but `positif(neg)` échoue et il n'y a pas besoin de rechercher explicitement des informations sur les instances négatives.
- les attributs des nœuds sont représentés par des faits Prolog (par exemple, `area(g1_n1,108079)`) tout comme les arêtes (par exemple : `touches(g3_n7,g3_n1)`).

Un exemple est donné avec la figure 7 et le code 2.

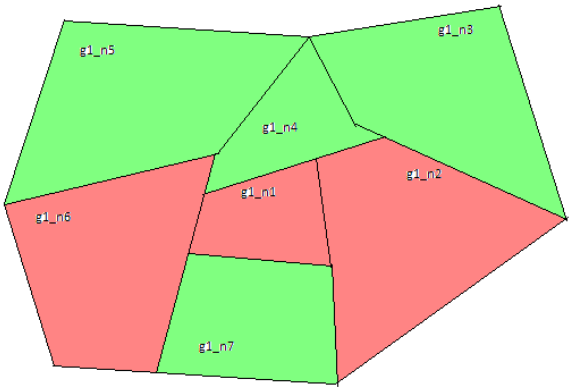


FIGURE 7 – Exemple de zone à représenter en Prolog

Code 2 – Extrait de Prolog généré pour la figure 7

```
instance(g1).
positif(g1).
inzone(g1,g1_n1).
inzone(g1,g1_n2).
inzone(g1,g1_n3).
inzone(g1,g1_n4).
inzone(g1,g1_n5).
inzone(g1,g1_n6).
inzone(g1,g1_n7).
touches(g1_n1,g1_n2).
touches(g1_n2,g1_n1).
touches(g1_n1,g1_n4).
touches(g1_n4,g1_n1).
touches(g1_n1,g1_n6).
touches(g1_n6,g1_n1).
touches(g1_n1,g1_n7).
touches(g1_n7,g1_n1).
touches(g1_n2,g1_n3).
touches(g1_n3,g1_n2).
[...]
cdistance(g1_n1, g1_n2, 123).
cdistance(g1_n2, g1_n1, 123).
[...]
area(g1_n1, 45).
[...]
perimeter(g1_n1, 112).
[...]
land_use(g1_n1,1).// 1 = sol artificiel
land_use(g1_n2,1).
land_use(g1_n3,3).// 3 = prairie
```

3.2 Inférence de règles

Suite à l'étude bibliographique, il a été décidé de concevoir une application qui génère des règles classant les exemples. Il convient de définir quelles sont les règles qui peuvent être produites, comment celles-ci se composent avant d'expliquer comment elles sont effectivement générées.

3.2.1 Définitions

Une règle est un ensemble de littéraux (cf. annexe A).

- R_2 est une **extension** d'une règle R_1 si R_2 s'écrit `R1, L1, ..., Ln` où chaque L_i désigne un littéral. On appellera **ancêtre** la relation inverse. R_2 est une **fille** de R_1 si R_2 n'a qu'un littéral de plus que R_1. Par exemple, dans la notation de Prolog `instance(G), inzone(G,X0)` est une fille de `instance(G)`

– Un **tuple** est une substitution de valeurs à des variables.
– Le **gain** (*FOIL gain*) d'une extension R' de R se calcule de la manière suivante :

$$gain(R, R') = s \times \left(\log \frac{p_{R'}}{t_{R'}} - \log \frac{p_R}{t_R} \right)$$

où $t_{R/R'}$ désigne le nombre de tuples qui satisfont R/R', $p_{R/R'}$ le nombre de tuples correspondant à des instances positives qui les satisfont, s le nombre de tuples qui satisfont R et qui peuvent être étendus en des tuples qui satisfont R'.

3.2.2 Algorithme d'inférence de règles

Les règles sont générées de manière incrémentale, littéral par littéral. Le choix du littéral à ajouter à la règle en cours se fait par génération des littéraux possibles, selon les prédicats disponibles (P dans l'algorithme 2) et les variables existantes dans la règle en cours, puis l'un d'eux est sélectionné selon le gain apporté. Afin de caractériser les instances, au moins l'un des paramètres d'un nouveau littéral est une variable déjà utilisée dans un des littéraux de la règle.

En Prolog, l'ordre des littéraux dans la conjonction importe : le système cherchera à trouver des valeurs pour les variables d'abord pour satisfaire le premier littéral de la règle avant de satisfaire la suite de la règle de la même manière. L'objectif est d'abord de trouver des règles pour caractériser les instances. Ainsi, toute règle commence par le littéral `instance` appliqué sur une variable (ligne 3 de l'algorithme 2).

Au cours de l'algorithme, on manipule un ensemble E qui contient l'ensemble des séquences de termes qui peuvent être paramètres du littéral à ajouter en queue de règle : chacune de ces séquences de termes contient au moins une variable qui apparait déjà dans la règle, les autres éléments pouvant être des variables déjà présentes ou non dans la règle et des constantes. Si n est l'arité maximale des prédicats, les séquences de E contiennent 1 à n termes. E est initialisé (ligne 5) avec la séquence de longueur 1 $\{G\}$ et des séquences contenant G et des constantes. L'opération en ligne 6 à augmenter E des tuples contenant des variables que l'on peut introduire. Lors de la première extension, la séquence de taille n $\{G, X_0, \ldots, X_{n-1}\}$ (l'ordre importe, ce sont des paramètres de fonctions booléennes) est notamment ajoutée. Cet ensemble est augmenté à chaque fois qu'au moins une nouvelle

variable est effectivement introduite dans la règle.

Algorithme 2: Inférence de règles

Données : Exemples positifs et négatifs de la base de connaissances
Entrées : Ensemble de prédicats P (arité maximale notée n)
Résultat : Ensemble S de règles

1 $S \leftarrow \varnothing$
2 **tant que** *les exemples positifs ne sont pas tous couverts* **faire**
3 $r \leftarrow$ nouvelle règle initialisée avec le littéral $instance(G)$
4 Initialisation de E `// ensemble de suites de 1 à` n `termes, paramètres potentiels des prédicats`
5 **tant que** *(longueur(r)* $\leqslant$ *seuil) ou (seules des instances positives vérifient r)* **faire**
6 On ajoute à E les éventuelles nouvelles suites de termes
7 Génération des littéraux possibles à l'aide de P et E
8 **pour tous les** *littéraux* l **faire**
9 $r' \leftarrow r \wedge l$
10 Évaluation de r' `// calcul du gain`
11 **fin**
12 Sélection de la règle r_{best}
13 $r \leftarrow r_{best}$
14 **fin**
15 Réduction de r
16 Retirer les exemples positifs qui vérifient r des exemples à traiter pour les éventuelles autres règles
17 Ajouter r à S
18 **fin**

La sélection de r_{best} (ligne 12) se fait par la sélection du meilleur gain. Néanmoins, si les tuples vérifiant la grand-mère de cette règle sont tous extensibles en tuples vérifiant cette règle, elle ne sera pas sélectionnée (cela signifie qu'on ajouterait deux fois de suite des littéraux n'opérant aucune sélection).

P peut contenir des prédicats-agrégats.

La réduction de règle (ligne 15) consiste à enlever les derniers littéraux tant que le gain qu'ils apportent est nul ou que tous les tuples de la règle parente ont une extension permettant de vérifier la règle avec le prédicat supplémentaire.

3.3 Outils et implémentation

Afin de pouvoir mettre en place l'architecture parallèle de l'algorithme, l'un des langages choisis est C++ en sa version C++11, munie d'une bibliothèque thread permettant une implémentation efficace simplement.

Afin de gérer la partie logique du travail, le langage Prolog intervient éga-

lement naturellement. Il existe plusieurs implémentations de Prolog. Devant le besoin d'inclure du code Prolog à l'intérieur du code C++, l'implémentation choisie devait permettre une interface entre C++ et Prolog. Trois outils, SICStus Prolog, SWI-Prolog et YAP (*Yet Another Prolog*), ont été testés. Les critères de choix ont été le respect d'une norme Prolog (ISO-Prolog, que les trois implémentations testées reconnaissent), les performances du système, la disponibilité (licence publique/privée) et l'interfaçage avec le C++[10]. Ainsi, pour ces trois derniers points, le choix s'est porté sur YAP.

Malheureusement, YAP supporte mal d'être utilisé en parallèle. Il n'y a donc pas, au jour de la date de rendu de ce rapport, d'implémentation parallèle fonctionnelle. Néanmoins, l'implémentation série est fonctionnelle.

Les agrégats n'ont pas encore été effectivement implémentés. Un moyen de les considérer serait de les définir dans Prolog à l'aide du prédicat `setof` qui permet d'obtenir les substitutions d'un ensemble de variables précisées qui permettent de vérifier une règle. L'utiliser permet de réaliser un agrégat de variables sur lesquelles on peut faire diverses opérations. Le code 3 montre un moyen de définir le prédicat *distance moyenne* : définition de la moyenne d'une liste (lignes 1–3), redéfinition du prédicat permettant d'obtenir le n^e élément d'une liste (lignes 5–6), définition d'un prédicat permettant d'obtenir la liste des n^{es} éléments des listes d'une liste de listes (lignes 8–9) et (enfin !) définition du prédicat de distance moyenne.

Afin de gérer les instances qu'il faut encore considérer lors de la construction de nouvelles règles, à l'initialisation de notre programme, un fait `do(gX)` est enregistré pour chaque instance gX. Pour retirer les exemples positifs des exemples à traiter (ligne 16 de l'algorithme 2) après la finalisation d'une règle r, le fait `do(gX)` est annulé pour toutes les instances positives qui vérifient r.

Notre programme ne garde pas en mémoire l'ensemble des tuples qui vérifient une règle r mais garde néanmoins trace des instances qui vérifient r. Chaque règle est identifiée par un indice qui lui est propre. À chaque fois qu'il est établi qu'une instance g_i vérifie la règle r_j d'indice j, le fait `checkrule(gi,rj)` est enregistré. Le prédicat $checkrule(G, r_j)$, placé en tête d'une règle fille de r_j, permet au système Prolog d'unifier l'identifiant de sous-graphe G uniquement avec des identifiants de sous-graphe qui vérifient r_j. Ce prédicat est également utilisé au point précédent pour rechercher directement les instances positives qui vérifient la règle nouvellement créée.

Code 3 – Définition du prédicat-agrégat *distance moyenne*

```
1 avg_aux([A],1,A).
2 avg_aux([A|L],N,V):-avg_aux(L,M,Vl),N is M+1, V is (M*
    Vl+A)/N.
3 avg(L,V) :- avg_aux(L,_,V).
4
5 nth(1,[A|_],A).
6 nth(N,[_|L],A) :- M is N-1, nth(M,L,A).
```

```
7
8  lnth(_,[],[]).
9  lnth(N,[La|L],[A|R]) :- nth(N,La,A), lnth(N,L,R).
10
11 avdistance(Parcel,Avg) :- setof([OtherParcel,D],(
       cdistance(Parcel,OtherParcel,D)),L), lnth(2,L,Ld),
       avg(Ld,Avg).
```

4 Expérimentations

4.1 Conditions

Lors des tests, les prédicats suivants constituent l'ensemble P de l'algorithme 2 :

- `inzone/2` : $inzone(G, X)$ signifie que la parcelle X est dans l'instance G
- `touches/2` : $touches(A, B)$ signifie que les parcelles A et B sont adjacentes
- `area/2` : $area(X, A)$ signifie que l'aire de la parcelle X vaut A
- `land_use/2` : $land\_use(X, N)$ décrit l'occupation du sol de la parcelle X avec l'équivalence pour N :
  1. sol artificiel
  2. zones agricoles
  3. végétation semi-naturelle/prairie
  4. eau
  5. forêt de conifères
  6. forêt de feuillus
  7. kaolin/sol nu clair
  8. sol nu
  9. estran
  10. plage
  11. voie de communication
- `perimeter/2` : $perimeter(X, P)$ signifie que le périmètre de la parcelle X vaut P
- `cdistance/3` : $cdistance(X, Y, D)$ signifie que la distance entre les centroïdes des parcelles X et Y vaut D.
- `</2` : $<(A, B)$ si $A < B$

Les constantes utilisées dans l'ensemble E de l'algorithme 2 sont les entiers de 0 à 50.

4.2 Quelques règles

Avec les prédicats de 4.1, des règles peuvent être générées à partir d'un jeu d'exemples. Néanmoins, les résultats obtenus ne sont pas très satisfaisants dans l'ensemble. Une règle générée couvre souvent un seul ou deux exemples positifs. Les dernières règles générées couvrent parfois beaucoup d'exemples négatifs. Par exemple, pour un jeu de 200 instances, dont 70 positives, 41 règles ont été générées et la dernière règle couvrait 70 instances négatives pour 6 instances positives alors non couvertes par les règles précédentes.

`instance(G), inzone(G, X0), land_use(X0, 3), cdistance(X0, X4, 21), land_use(X4, 3)` ne vérifiait qu'un exemple positif. Cette règle signifie que le sous-graphe comporte deux parcelles de prairies dont les centroïdes sont séparés de 21 (unités arbitraires).

`instance(G), inzone(G, X0), land_use(X0, 3), cdistance(X0, X4, 31), touches(X0, X4), land_use(X4, 1)` vérifiait deux exemples positifs. Cette règle signifie que le sous-graphe comporte une parcelle de prairie adjacente à une parcelle urbaine, et les centroïdes de ces deux parcelles sont distants de 31.

Quelques améliorations sont possibles pour obtenir des résultats satisfaisants. Plusieurs attributs mentionnés en section 3 (la parcelle est-elle une école ? un centre commercial ? un parc naturel ?) pourraient être utilisés à l'aide de prédicats adaptés. Aucun agrégat n'a encore été introduit. L'ensemble E de l'algorithme 2 est très général, on peut tenter d'avoir un ensemble de suite de termes admissibles propre à chaque littéral. Les constantes ne devraient pas être statiques mais dépendre des données (par exemple, lorsqu'un prédicat, comme `perimeter` introduit une variable numérique, analyser les différentes valeurs qui peuvent être prises en fonction de la classe de l'instance). $<$ n'est jamais sélectionné. On peut former des littéraux doubles en associant des prédicats introduisant des variables numériques et le prédicat de comparaison $<$. Les contraintes empêchant l'éventuelle sélection d'une règle sont peut-être trop fortes et méritent peut-être d'être un peu détendues (par exemple, autoriser que les tuples vérifiant une règle r_{gm} grand-mère d'une règle r sont extensibles en tuples vérifiant r, mais l'interdire pour la règle arrière-grand-mère).

On remarque également que l'opération de réduction de règle est à affiner : s'occuper uniquement de la queue de la règle n'est pas suffisant. Par exemple, dans `instance(G), inzone(G, X0), land_use(X0, 1), cdistance(X0, X6, 43), land_use(X6, 1), cdistance(X0, X10, 30), cdistance(X10, X6, X12), land_use(X10, 1)`, le littéral `cdistance(X10, X6, X12` n'apporte pas d'information.

4.3 Quelques chiffres

L'utilisation mémoire et le temps d'exécution ont été testés pour les conditions de 4.1, de 10 à 200 instances dans le jeu d'exemples (par pas de de 10) et un seuil de distance maximale entre les centroïdes des parcelles et le centroïde de la parcelle centrale compris entre 500 et 4000 (par pas de 500) (plus le seuil est élevé, plus la quantité d'information d'un exemple sera conséquente). Une dizaine d'heures de calcul sur un ordinateur disposant de 4Go de mémoire vive et d'un processeur cadencé à 2GHz ont été nécessaires afin d'obtenir ces mesures. Pour des raisons de lisibilités, seules quatre courbes sont présentes sur chaque graphique.

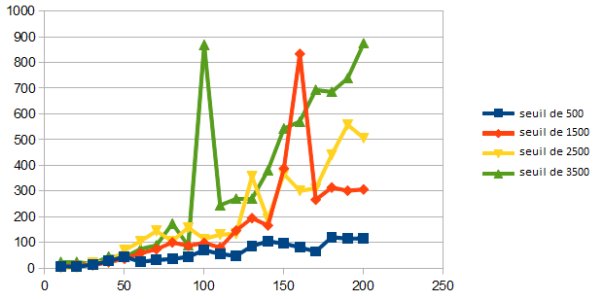


FIGURE 8 – Temps d'exécution (en secondes) pour quatre valeurs de seuils en fonction du nombre d'instances

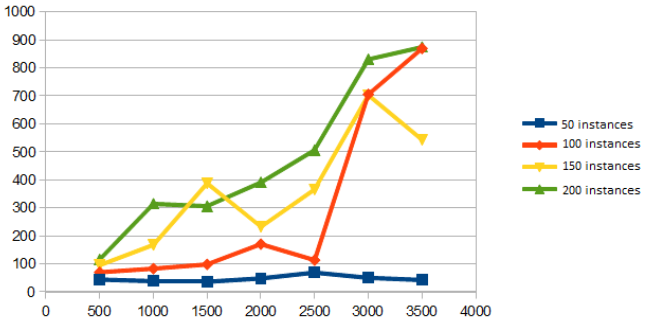


FIGURE 9 – Temps d'exécution (en secondes) pour quatre valeurs de nombre d'instances en fonction du seuil

On remarque sans surprise d'après les figures 8 et 9 que plus il y a d'exemples, et plus les exemples sont complexes, alors plus le temps mis par le générateur de règles pour terminer est important. On remarque néanmoins quelques irrégularités (comme un pic à 100 instances pour la courbe de seuil 3500) qui indiquent que la composition même des exemples peut influencer

sur le temps de calcul. Pour des petits jeux d'exemples (de 50 instances sur la figure 9), le seuil semble ne pas trop influencer le temps d'exécution.

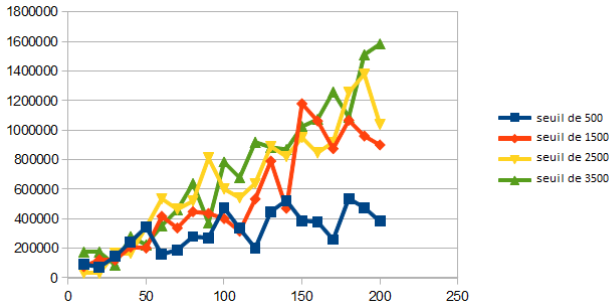


FIGURE 10 – Utilisation mémoire (en kilooctets) pour quatre valeurs de seuils en fonction du nombre d'instances

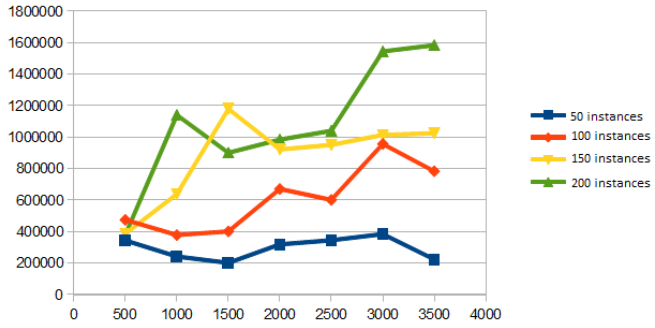


FIGURE 11 – Utilisation mémoire (en kilooctets) pour quatre valeurs de nombre d'instances en fonction du seuil

Pour l'occupation mémoire, on observe des phénomènes similaires sur les figures 10 et 11. Le nombre d'instance est très influent, le seuil beaucoup moins. Encore une fois, le seuil ne semble pas altérer l'occupation mémoire pour les petits jeux de 50 exemples.

Le paramètre le plus influent sur les performances de l'algorithme (en termes de temps de calcul et de mémoire utilisée) est le nombre d'instances. De nombreuses améliorations (cf. 4.2) restent à faire pour qu'il soit intéressant d'utiliser l'algorithme sur un grand jeu d'exemples.

5 Conclusion

Les paysages sont des organisations spatiales d'éléments paysagers : ils peuvent être représentés par des graphes ou par la liste des relations qui les décrit. Lors de ce stage, une approche basée sur la PLI a été choisie afin de répondre à un problème d'apprentissage supervisé : peut-on caractériser les paysages qui vont subir une urbanisation dans l'avenir ? Ce choix a été fait en fonction des critères d'expressivité et d'interprétabilité du modèle. Un algorithme proche de Foil a été implémenté afin de générer un ensemble de règles répondant à la problématique sur des jeux d'exemples obtenus grâce à une analyse diachronique de cartes de Lorient entre 2000 et 2010. Ces cartes ont subi différents traitements (analyse de la composition du sol, segmentation en parcelles, segmentation en sous-graphe et conversion en programme Prolog) pour servir de jeux d'exemples.

Malgré des performances techniques satisfaisantes (temps de calcul et occupation mémoire), l'implémentation courante produit par contre des résultats peu expressifs à cause du peu de prédicats que l'on admet dans les règles. Il manque notamment les prédicats-agrégats qui résument des situations, par exemple en imposant un minorant d'un ensemble de distance. La gestion des prédicats s'appliquant sur des paramètres numériques est à améliorer. Il faut encore trouver un moyen de paralléliser l'implémentation malgré le support de Prolog choisi (YAP) afin d'avoir des temps de calculs encore meilleurs.

Références

[1] L. Miclet A. Cornuéjols. *Apprentissage Artificiel : Concepts et algorithmes*. Eyrolles, 2002.

[2] A. Appice, M. Ceci, A. Lanza, F. Lisi, and D. Malerba. Discovery of spatial association rules in geo-referenced census data : A relational mining approach. *Intell. Data Anal.*, 7(6) :541–566, December 2003.

[3] A. Appice, A. Ciampi, A. Lanza, D. Malerba, A. Rapolla, and L. Vetturi. Geographic knowledge discovery in ingens : An inductive database perspective. In *Proceedings of the 2008 IEEE International Conference on Data Mining Workshops*, ICDMW '08, pages 326–331. IEEE Computer Society, 2008.

[4] H. Arimura, T. Uno, and S. Shimozono. Time and space efficient discovery of maximal geometric graphs. In *Proceedings of the 10th international conference on Discovery science*, DS'07, pages 42–55. Springer-Verlag, 2007.

[5] A. Atramentov, H. Leiva, and V. Honavar. A multi-relational decision tree learning algorithm - implementation and experiments. In *Proceedings of the 13th International Conference on Inductive Logic Programming (ILP 2003)*, pages 38–56. Springer-Verlag, 2003.

[6] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artif. Intell.*, 101(1-2) :285–297, 1998.

[7] K. M. Borgwardt. *Graph Kernels*. PhD thesis, Ludwig-Maximilians-Universität München, 2007.

[8] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1) :47–56, January 2005.

[9] N. Chelghoum, K. Zeitouni, T. Laugier, A. Fiandrino, and L. Loubersac. Fouille de données spatiales. approche basée sur la programmation logique inductive. In *EGC*, pages 529–540, 2006.

[10] B. Demoen and P.-L. Nguyen. Odd Prolog benchmarking. CW Reports CW312, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2001.

[11] A. Despres. Fouille d'un grand graphe pour des données géospatiales, 2012. Rapport de stage INSA.

[12] W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. Yu, and O. Verscheure. Direct mining of discriminative and essential frequent patterns via model-based search tree. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 230–238. ACM, 2008.

[13] R. Frank, M. Ester, and A. Knobbe. A multi-relational approach to spatial classification. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 309–318. ACM, 2009.

[14] Warodom Geamsakul, Takashi Matsuda, Tetsuya Yoshida, Hiroshi Motoda, and Takashi Washio. Classifier construction by graph-based induction for graph-structured data. In K-Y Whang, J. Jeon, K. Shim, and J. Srivastava, editors, *Advances in Knowledge Discovery and Data Mining*, volume 2637 of *Lecture Notes in Computer Science*, pages 52–62. Springer Berlin Heidelberg, 2003.

[15] H. Guo and H. L. Viktor. Multirelational classification : a multiple view approach. *Knowl. Inf. Syst.*, 17(3) :287–312, 2008.

[16] J.-F. Guo, J. Li, and W.-F. Bian. An efficient relational decision tree classification algorithm. In *Proceedings of the Third International Conference on Natural Computation - Volume 03*, ICNC '07, pages 530–534. IEEE Computer Society, 2007.

[17] T. Guyet. Fouille de données spatiales pour la caractérisation spatiale de paysages en lien avec des fonctionnalités agro-écologiques. In *Spatial Analysis and GEOmatics (SAGEO'10)*, page 3, 2010.

[18] J. Huan, W. Wang, A. Washington, J. Prins, R. Shah, and A. Tropsha. Accurate classification of protein structural families using coherent subgraph analysis. In *In Proc. Pacific Symposium on Biocomputing*, pages 411–422, 2004.

[19] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '00, pages 13–23. Springer-Verlag, 2000.

[20] M. Kuramochi and G. Karypis. Discovering frequent geometric subgraphs. *Inf. Syst.*, 32(8) :1101–1120, December 2007.

[21] F. A. Lisi and D. Malerba. Inducing multi-level association rules from multiple relations. *Mach. Learn.*, 55(2) :175–210, May 2004.

[22] D. Malerba. A relational perspective on spatial data mining. *Int. J. Data Mining, Modelling and Management*, 1(1) :103–118, 2008.

[23] D. Malerba and F. A. Lisi. An ILP method for spatial association rule mining. In *In Working notes of the First Workshop on Multi-Relational Data Mining*, pages 18–29, 2001.

[24] S. Muggleton and L. De Raedt. Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19(20) :629–679, 1994.

[25] PhuChien Nguyen, Kouzou Ohara, Akira Mogi, Hiroshi Motoda, and Takashi Washio. Constructing decision trees for graph-structured data by chunkingless graph-based induction. In W-K Ng, M. Kitsuregawa,

J. Li, and K. Chang, editors, *Advances in Knowledge Discovery and Data Mining*, volume 3918 of *Lecture Notes in Computer Science*, pages 390–399. Springer Berlin Heidelberg, 2006.

[26] J. R. Quinlan and R. M. Cameron-jones. Foil : A midterm report. In *In Proceedings of the European Conference on Machine Learning*, pages 3–20. Springer-Verlag, 1993.

[27] N. Selmaoui-Folcher and title = How to use "classical" tree mining algorithms to find complex spatio-temporal patterns ? booktitle = Proceedings of the 22nd international conference on Database and expert systems applications - Volume Part II series = DEXA'11 year = 2011 isbn = 978-3-642-23090-5 location = Toulouse, France pages = 107–117 numpages = 11 url = http ://dl.acm.org/citation.cfm ?id=2033546.2033558 acmid = 2033558 publisher = Springer-Verlag keywords = complex spatio-temporal trees, data pre-processing, spatio-temporal data mining, unordered tree mining algorithms Flouvat, F.

[28] N. Shervashidze, SVN Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient Graphlet Kernels for Large Graph Comparison. In *12th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Society for Artificial Intelligence and Statistics, 2009.

[29] A. Thakkar and Y. P. Kosta. Survey of multi relational classification (MRC) approaches & current research challenges in the field of MRC based on multi-view learning. *International Journal of Soft Computing & Engineering*, 1(6) :247–252, 2012.

[30] Dr. M. Thangaraj and C. R. Vijayalakshmi. Article : A study on classification approaches across multiple database relations. *International Journal of Computer Applications*, 12(12) :1–6, January 2011. Published by Foundation of Computer Science.

[31] R. Trépos. *Apprentissage symbolique à partir de données issues de simulation pour l'aide à la décision : gestion d'un bassin versant pour une meilleure qualité de l'eau.* PhD thesis, Rennes 1, 2008.

[32] X. Yin, J. Han, J. Yang, and P. Yu. Crossmine : Efficient classification across multiple database relations. In *In Proc. 2004 Int. Conf. on Data Engineering (ICDE'04), Boston,MA*, 2004.

A Quelques rappels de logique pour la PLI

A.1 Définitions

- Un **prédicat logique** est une fonction booléenne (qui renvoie *vrai* ou *faux*).
- Un **terme** peut être une constante simple (nombre entier, nombre flottant, type énuméré), une variable ou un terme composé d'une fonction sur des termes (ces termes composés étant inutilisés dans notre application, ils ne sont pas considérés dans la suite).
- Un **littéral** est l'application d'un prédicat à des termes.
- Une **règle** est une conjonction de littéraux.

A.2 Conventions de Prolog

La PLI fonctionne sur l'idée d'une base de connaissances permettant de déterminer si un littéral peut être vrai. Ainsi, en Prolog, sont déclarés des faits et des implications de faits par des règles.
Exemples :

```
pred0(c0,1).
pred0(c1,0).
pred0(X).
pred1(X,Y) :- pred0(Y), pred0(Y,X).
```

Ceci signifie en formules logiques :

$$pred_0(c_0, 1)$$

$$\forall X \; pred_0(X)$$

$$\forall X, Y \left(pred_0(Y) \wedge pred_0(Y, X)\right) \Rightarrow pred_1(X, Y)$$

De manière générale, une règle Prolog `pred0(X1, ..., Xn) :- pred1(X1, ..., Xm), ..., predl(X1, ..., Xm)` équivaut dans le langage de la logique du premier ordre à :

$$\forall X_1, \ldots, X_m \; \left(\bigwedge_{i=1}^{l} pred_i(X_1, \ldots, X_m) \right) \Rightarrow pred_0(X_1, \ldots, X_n)$$

En Prolog, une `Variable` commence par une majuscule. Les autres termes (fonctions et constantes d'un type énuméré) et les noms de prédicat commencent par des minuscules.

Le modèle d'exécution d'un programme Prolog repose sur la méthode de démonstration automatique appelée résolution. Partant d'une clause but, il s'agit de "résoudre" pas à pas cette clause en la résolvant (réécrivant) au moyen des règles du programme. Si l'exécution parvient à la clause vide, le but est démontré.

La résolution de règles passe par une résolution littéral par littéral dans l'ordre d'écriture, de gauche à droite. Pour résoudre un littéral, le système cherche à unifier ses termes avec ceux d'un fait ou avec ceux du membre de gauche d'une règle d'implication. Dans le second cas, si l'unification est possible, le système doit résoudre les littéraux composant le membre de droite de la règle d'implication.

L'unification de deux termes t_1 et t_2 est le fait de trouver un troisième terme t tel qu'en substituant des valeurs de termes à des variables (éventuellement des variables) de t_1 et t_2, $t'_1 = t'_2 = t$. t est un unificateur de t_1 et t_2

Exemples :

- `predicat(a)` et `predicat(a)` : `a` et et `b` sont des constantes différentes, il n'existe pas de substitution de variable les unifiant.
- `predicat(X)` et `predicat(Y)` : `X` et `Y` sont deux variables que l'on peut unifier en substituant (par exemple) `Y` à `X`
- `predicat(Z)` et `predicat(c)` : `Z` et `c` sont unifiables en substituant `c` à `Z`

La résolution de règle peut être utilisée aussi bien pour savoir si une règle peut être vérifiée mais aussi pour récupérer tous les tuples qui correspondent à des valuations pour lesquelles la règle est vérifiée.