Institut National
Polytechnique de Lorraine

Département de formation doctorale en informatique

École doctorale IAEM Lorraine

Reconnaissance de symboles sans connaissance a priori

THÈSE

présentée et soutenue publiquement le 6 novembre 2006

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine
(spécialité informatique)

par

Daniel Zuwala

Composition du jury

Rapporteurs :	Jean-Marc OGIER
	Laurent HEUTTE
Examinateurs :	Karl TOMBRE
	Salvatore TABBONE
	Kamel SMAILI
	Jean-Yves RAMEL

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Remerciements

Je tiens à remercier les personnes qui me font l'honneur de participer au jury de cette thèse. Je remercie en particulier Jean-Marc Ogier et Laurent Heutte pour l'intérêt qu'ils ont porté à ce travail en acceptant d'en être les rapporteurs. Je remercie également Kamel Smaili et Jean-Yves Ramel pour avoir accepté d'être examinateur de cette thèse.

Mes remerciements s'adressent également à Karl Tombre qui a été mon directeur de thèse pendant les trois années passés. Il a su orienter mon travail dans les bonnes directions tout en me laissant une large autonomie.

Je tiens à remercier tout particulièrement Salvatore Tabbone, qui a été l'encadrant principal ainsi que co-directeur de ce travail pendant trois ans. Ce travail est en grande partie dû à ses conseils ainsi que sa ténacité à exploiter jusqu'au bout les idées. Je le remercie également pour son gros travail de relecture dans tout ce que j'ai pu rédigé et en particulier cette thèse.

Je remercie également toute l'équipe QGAR qui m'a accueilli. En particulier merci au trio Philippe Dosch, Gérald Masini et Jan Rendek pour la patience qu'ils ont eu envers moi et mes multiples questions en C++. Merci également à Laurent Wendling mon ex-colocataire de bureau pour sa bonne humeur. Merci aussi à ceux qui sont encore entrain de trimer en thèse pour la bonne ambiance qui s'est dégagé dans le bureau des thèsards (Sabine, JP et Oanh).

Merci également à tous ceux que j'oublie mais qui d'une manière ou d'une autre m'ont permis de passer de bons moments.

Table des matières

Introduction

La reconnaissance de symboles dans des documents graphiques est un vaste champ de recherche. En effet, les besoins sont énormes en raison justement des grandes quantités de documents à traiter et de leur variété : cartes, schémas électriques, plan architecturaux (cf Fig 1). Cette diversité est le talon d'Achille des méthodes de reconnaissance de symboles car il est bien souvent impossible d'avoir une méthode qui fonctionne sur tous les types de documents et sur tous les types de symboles que l'on peut rencontrer. Bien souvent, les méthodes seront destinées à un certain type de document et ne pourront traiter qu'un nombre restreint et bien souvent prédéfini de symboles.

Les techniques développées dans ce domaine s'articulent souvent en trois phases : une phase de représentation, une phase de description et une phase de classification [CV00].

La phase de représentation a pour objectif de réduire la quantité d'informations présentes dans le document afin de mettre en avant les informations susceptibles de nous intéresser. Cette phase est aussi le moment pour tenter de s'affranchir au maximum du bruit qui peut être présent. C'est dans cette phase qu'ont lieu tous les traitements appelés de bas niveau qui ont fait et font toujours l'objet de larges investigations. On y retrouve (liste non exhaustive) :

- la binarisation,
- la squelettisation
- les approximations polygonales,
- la vectorisation,

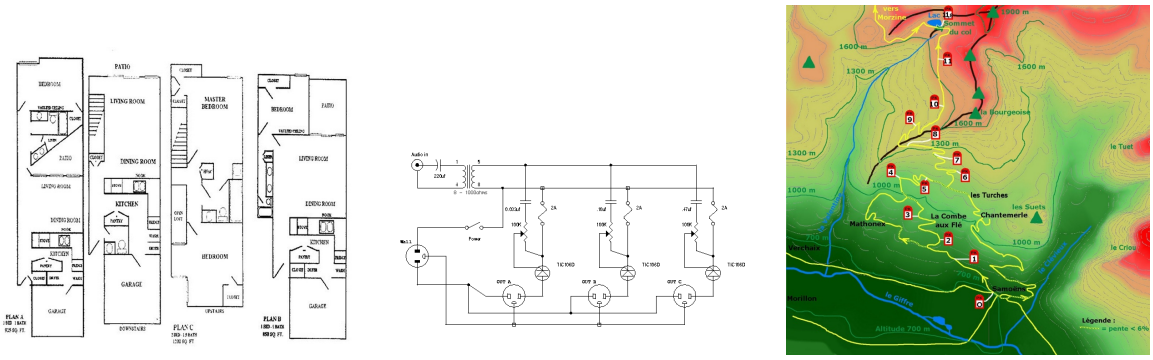


FIG. 1 – Différents types de documents qui peuvent nécessiter une reconnaissance de symbole.

– les opérations morphologiques mathématiques.

Une fois la phase de représentation achevée, on isole les symboles du reste du document. Cette étape très délicate demeure encore un des points durs qui suscite de nombreux travaux.

La phase de description permet de passer de l'image qu'on a du symbole dans le document à une description qui soit plus facilement utilisable. Ici, on peut identifier deux types de descriptions différentes : celles qui sont statistiques et qui vont produire un vecteur de caractéristique de dimensions plus ou moins grandes, et celles qui sont structurelles et qui vont décrire le symbole sous la forme d'un graphe. Le choix de la description est cruciale. En effet, la description idéale doit être stable, concise, unique, accessible et invariante aux transformations affines. Nous reviendrons plus en détails sur les différentes descriptions.

La phase de classification permet la reconnaissance proprement dite des symboles. Avoir une description des symboles ne suffit pas, il faut ensuite être capable de comparer ces descriptions aux descriptions que l'on peut avoir des symboles que nous recherchons. On a là aussi tout un éventail de possibilités selon, bien entendu, la description choisie.

Ainsi d'un côté, nous avons les classifieurs statistiques qui comprennent entre autres les plus proches voisins, les réseaux de neurones, les réseaux bayésiens, etc. Et d'un autre côté, dans le cadre d'une description structurelle, nous avons à notre disposition l'éventail des méthodes permettant de faire du *graph matching* exact ou inexact.

Les stratégies utilisées

Il est ensuite possible d'utiliser principalement deux types de stratégies : une stratégie *bottom-up* ou une stratégie *top-down*. La stratégie *bottom-up* qui part du plus faible niveau de représentation pour ensuite évoluer vers des représentations de niveaux de plus en plus élevés jusqu'à ce qu'on puissent comparer les symboles candidats aux symboles prototypes. Cette stratégie est la plus largement répandue. La stratégie *top-down* part du symbole prototype et va essayer de faire correspondre ce modèle aux données. Ce type de stratégie est particulièrement judicieux quand on veut utiliser les connaissances que l'on possède a priori.

Plan de la thèse

Dans cette thèse nous nous proposons non pas d'ajouter aux méthodes pré-existantes une méthode supplémentaire qui pourrait satisfaire un nombre restreint de situations, mais nous proposons une méthode qui pourrait être générique à un ensemble élargi de documents en n'utilisant pas de connaissances a priori sur les symboles recherchés. Bien sûr, cette méthode aura certainement de moins bons résultats face à une méthode qui serait spécialement dédiée à un type précis de document. Mais, nous soutenons l'idée qu'elle peut permettre d'avoir des résultats corrects sur des documents de nature totalement différente. Nous verrons aussi que nous posons des hypothèses très souples quant à la nature des symboles que nous pouvons détecter puis reconnaître, ce qui nous permet d'avoir un système de reconnaissance de symbole sans connaissance a priori.

Cette thèse arbordera de manière successive les problèmes de représentation, de des-

cription et de classification. Nous commencerons par aborder les problèmes de description. Ainsi dans le premier chapitre nous nous attarderons à voir l'éventail des descripteurs existants dans la littérature. Ce sera l'occasion pour nous de présenter de manière plus spécifique des descripteurs qui auront attirés notre attention. Puis dans le chapitre 2, nous étudierons plus en détail ces descripteurs, ce qui nous amènera par la suite à faire une étude comparative de performances. Le chapitre 3 abordera le problème de la représentation. Nous y verrons comment à partir d'un document, nous pouvons extraire, sans connaissance a priori sur les symboles recherchés, des zones susceptibles de contenir des symboles qui seront ensuite identifiés à l'aide d'un descripteur. La méthode présentée générant de nombreux symboles candidats, et le descripteur choisit ayant une dimension élevé, nous devrons aborder l'épineux problème de la recherche des plus proches voisins dans le cadre de grandes bases de données possédant de grandes dimensions. Ainsi dans le chapitre 4, nous verrons les méthodes déjà existantes, tandis que dans le chapitre 5, nous y développerons notre propre méthode.

Première partie

A propos des descripteurs

Chapitre 1

Etat de l'art des descripteurs utilisés dans la reconnaissance de symboles

Dans ce chapitre, nous développons davantage la phase de description dont nous avons parlé en introduction. Il existe de nombreuses descriptions possibles et en faire une énumération exhaustive serait fastidieux. Nous proposons plutôt ici de décrire les méthodes qui permettront d'avoir un aperçu de ce vaste champ de recherches et que nous avons en partie expérimenté. Il existe plusieurs méthodologies qui permettent de classer toutes ces méthodes. Certaines divisent ces méthodes en fonction de leur aspect local ou global [AOCG01], d'autres les divisent en fonction de l'information qui est directement exploitée pour construire le modèle (image en niveau de gris, image binaire, contour ou squelette) [TJT96]. D'autres encore considèrent d'un côté les méthodes ayant un aspect statistique et de l'autre les méthodes ayant un aspect structurel [LVSM01]. Nous avons choisi cette dernière méthodologie.

1.1 Descripteurs syntaxiques et structurelles

Les méthodes structurelles et syntaxiques représentent les symboles en utilisant un ensemble adapté de primitives géométriques et de relations entre celles-ci. A chaque symbole correspond un modèle idéale. Un symbole candidat appartient à telle ou telle classe de symbole, quand sa représentation est suffisament proche de la représentation d'un des modèles. Les primitives les plus usités sont les lignes et les arcs. Une vectorisation doit donc être utilisée, ce qui introduit va souvent introduire du bruit et des déformations. Ainsi les méthode d'appariement entre les représentations doivent être tolérante aux erreurs.

1.1.1 Les grammaires

C'est en théorie des langages que les grammaires ont été initialement développées [Cho65]. Ces travaux ont ensuite été adaptés pour la reconnaissance de forme par Fu [Fu74]. Mais l'utilisation des grammaires était limité dans ce champ d'application. En effet, les

grammaires décrivent les éléments de manière linéaire, ne permettant que la concaténation d'éléments à droite et à gauche.

C'est pour pallier ces problèmes que Shaw [Sha69] a développé un langage PDL permettant de considérer la nature bi-dimensionnelle des images. Il définit quatre opérateurs qui permettent de combiner des éléments entre eux, afin de former des éléments plus complexes, ces éléments étant constitués de deux points particuliers appelés *tête* et *queue*.

Depuis, d'autres formes de grammaire ont vu le jour permettant, entre autre, de définir des formes encore plus complexes (comme dans le cas de l'analyse de partitions musicales [Coü96]). Notons aussi les plex-grammaires [Fed71] qui permettent de définir un nombre quelconque de points de connexions pour un élément, ainsi que les grammaires de graphes [Bun82] et [Fu83] qui permettent de combiner la richesse de la représentation des graphes avec la rigueur des grammaires.

1.1.2 Les graphes

Il est possible de représenter les symboles sous forme de graphes. La comparaison de deux graphes se fait alors à l'aide de méthodes dites de *graph matching* (nous en reparlerons dans la partie suivante). Il existe un nombre incalculable de représentations sous forme de graphe et nous n'en citons ici quelques unes [CFSV04].

Pour passer d'un document à sa représentation sous forme de graphe, le document subit plusieurs étapes dites de bas traitement. Il peut alors subir une squeletisation, une approximation polygonale, ou encore une vectorisation. C'est alors seulement qu'il peut être décrit sous la forme d'un graphe.

Les graphes relationel attribué sont les représentations les plus courantes au niveau structurel. Elles consistent à décrire le document à partir de structures que l'on a pu extraire (cela peut être des chaînes de points connectés, des segments, des arcs). On construit alors le graphe en regardant les relations que ces structures ont entre elles et en attribuant une valeur aux nœuds ou aux arcs du graphe qui rend compte de ces relations.

Ainsi Messmer [MB96], après avoir effectué une approximation polygonale du document construit un graphe dont les nœuds représentent les côtés du polygone avec comme attribut leur longueur, et dont les arcs représentent les relations connexes entre les segments avec comme attribut l'angle entre les côtés du polygone. Une autre manière de représenter le document est d'utiliser un graphe de région adjacentes qui consiste à trouver toutes les composantes connexes contenues dans le document, puis de construire un graphe qui tient compte des relations que ces composantes connexes ont entre elles [LLKM97].

1.2 Descripteurs pixels

On entend par descripteur pixels, les descripteurs qui se basent sur l'information photométrique des symboles. On peut par ailleurs encore subdiviser ces descripteurs en différentes catégories : ceux qui travaillent sur des caractéristiques géométriques, ceux qui

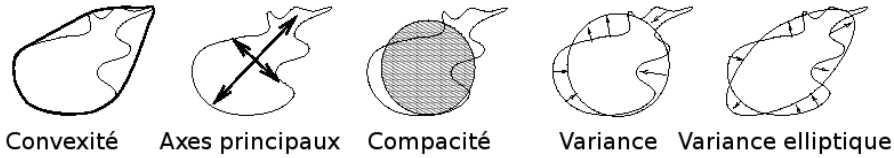


FIG. 1.1 – Exemples de descripteurs géométriques.

utilisent des moments géométriques et ceux qui appliquent une transformation à l'image avant d'en extraire des caractéristiques.

1.2.1 Caractéristiques géométriques

1.2.1.1 Descripteurs de forme simples

Il existe de nombreux descripteurs qui sont très facilement calculables à partir du contour de l'image (voir fig 1.1). On peut citer :

- l'aire.
- l'excentricité E :
$$E = \frac{\text{longueur de l'axe principal}}{\text{longueur de l'axe mineur}}$$
- l'orientation de l'axe principal.
- l'energie de courbure [YWB74] :
$$BE = \frac{1}{P}\int_0^P |K(p)|^2 dp$$
avec $K(p)$, la fonction de courbure, p la paramétrisation, et P la longueur totale de la courbe.
- la convexité C :
$$C = \frac{\text{Périmètre enveloppe convexe}}{\text{Périmètre contour}}$$
- la variance circulaire et elliptique : [PI97]

Ces descripteurs sont en général très rapides à calculer mais sont aussi peu discriminants. C'est pourquoi ils seraient à notre avis utiles pour faire un premier tri entre les symboles susceptibles d'être de bons candidats.

1.2.1.2 Les signatures basées sur le contour

Une signature représente une forme par une fonction 1D dérivée du contour. Il existe de nombreuses signatures [Zha02] comme :

- le profil centroidal,
- les coordonnées complexes,
- la distance au centroïd,

– l'angle tangent,
– l'angle cumulé,
– la courbure,
– l'aire,
– la longueur de la corde.

Ces signatures sont habituellement normalisées pour être invariantes à la translation et à l'échelle. L'invariance à la rotation peut être obtenue en faisant une permutation circulaire de la signature, ce qui vient alourdir le temps de calcul pour un appariement. L'avantage de ces signatures est globalement leur coût peu élevé en temps de calcul, mais en contrepartie, elles sont souvent assez sensibles au bruit, et le coût de l'appariement entre deux signatures peut être élevé.

1.2.1.3 Histogrammes de contraintes entre pixels

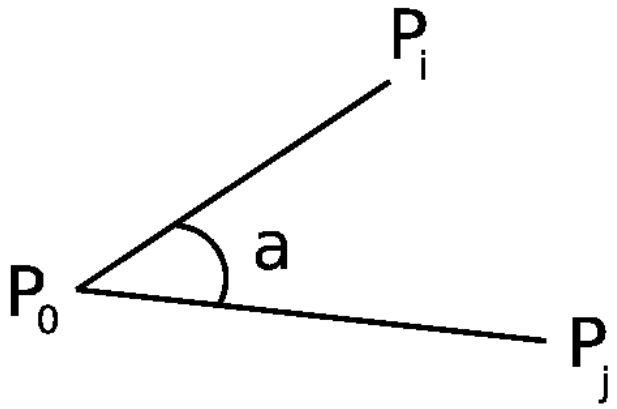


FIG. 1.2 –

Une autre approche récente et intéressante qui a attiré notre attention se base aussi sur la géométrie de la forme est celle proposée par Yang [Yan05]. Pour chaque point P_0 de la forme, Yang considère deux autres points P_i et P_j appartenant aussi à la forme. Pour ce triplet de point il est alors possible de calculer l'angle a, et le rapport minimum des distance L_{ij} qui sont définis par :

$$a = (P_0P_i, P_0P_j)$$

$$L_{ij} = min(\frac{P_0Pi}{P_0Pj}, \frac{P_0Pj}{P_0Pi})$$

A partir de ce point de référence, on peut examiner tous les voisins possibles, et l'on pourra alors construire deux histogrammes qui rendent compte de la distribution des angles et du rapport minimum des distances que l'on normalise par le nombre de couples de points.

On utilise ensuite chaque point de la forme comme point de référence, ce qui nous donne un ensemble d'histogrammes. On peut alors calculer un histogramme moyen pour les angles et pour le rapport minimum des distances, que l'on utilise comme descripteur.

Les résultats obtenus sont très intéressants, l'auteur affirmant obtenir un taux de reconnaissance de pratiquement 100% sur tous les tests GREC'03. Cependant la complexité en $O(n^3)$ rend ce descripteur peu adéquat dans le cadre de grandes bases de données. Pour réduire le coût en terme de calcul, il est plus judicieux d'appliquer ce descripteur sur le squelette ou sur le contour des formes recherchées. Les deux seuls paramètres à régler sont l'échantillonnage pour l'histogramme des angles et l'histogramme du rapport minimum des distances.

1.2.2 Moments géométriques

1.2.2.1 Moments géométriques invariants

Les moments géométriques introduits par Hu [Hu62] sont des descripteurs qui ont fait l'objet de larges investigations et qui sont toujours largement utilisés [DBM77], [CH87], [RPAK88], [ZS00], [QR99].

Soit deux entiers $p, q \in \mathbb{Z}$ et une image en niveau de gris $I(x, y)$, les moments géométriques se calculent de la manière suivante :

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(x, y) x^p y^q dx dy$$

Ces moments déterminent de manière unique l'image $I(x, y)$, c'est-à-dire qu'à une image $I(x, y)$ ne correspond qu'un unique ensemble de moments m_{pq} et réciproquement.

Pour être invariant à la translation, à la rotation ou à l'échelle, il est nécessaire de définir des moments centraux M_{pq} exprimés par rapport au centre de gravité de l'image de coordonnée (x_0, y_0). M_{pq} est alors défini par :

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(x, y)(x - x_0)^p (y - y_0)^q dx dy$$

L'invariance à l'échelle s'obtient en normalisant ces moments. On obtient alors les moment centraux normalisés μ_{pq} définis par :

$$\mu_{pq} = \frac{M_{pq}}{M_{00}^{\gamma}}$$

avec $\gamma = \frac{p+q}{2} + 1$

Hu [Hu62] montre alors qu'on peut obtenir l'invariance à la rotation en utilisant les 7 moments suivants :

$$HMI_1 = \mu_{02}\mu_{20}$$

$$HMI_2 = (\mu_{02} - \mu_{20})^2 + 4\mu_{11}^2$$

$$HMI_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2$$

$$HMI_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2$$

$$HMI_5 = (\mu_{30} - 3\mu_{12})\Big(\mu_{30} + \mu_{12})((\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2\Big) +$$

$$(3\mu_{21} - \mu_{30})\Big(\mu_{21} + \mu_{03})(3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2\Big)$$

$$HMI_6 = (\mu_{20} - \mu_{02})^2\Big((\mu_{30} + \mu_{12})^2 - (\mu_{21} - \mu_{03})^2\Big) + 4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03})$$

$$HMI_7 = (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12})\Big((\mu_{30} + \mu_{12})^2 - 3(\mu_{21} - \mu_{03})^2\Big) -$$

$$(\mu_{30} - 3\mu_{12})(\mu_{12} + \mu_{03})\Big(3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2\Big)$$

Les six premiers moments sont invariants à la translation, la rotation, l'échelle ainsi qu'aux reflexions. Le septième moment n'étant pas invariant aux réflexions, et permettant donc de différencier les images dites "miroirs".

1.2.2.2 Moments invariants de Zernike

Introduit par Teague [Tea79], les moments de Zernike ont eux aussi fait l'objet de larges investigations et sont aussi toujours largement utilisés [KH90], [KS02], [CRM03], [HN04]. Cette approche consiste à projeter l'image dans un espace vectoriel en utilisant un ensemble de polynômes orthogonaux (les polynômes de Zernike) qui sont définis comme ci dessous :

$$V_{nl}(x, y) = V_{nl}(\rho cos\theta, \rho sin\theta) = R_{nl}(\rho)e^{il\theta}$$

avec :

$$i^2 = -1, n \in \mathbb{N}, l \in \mathbb{Z}, |l| \leq n \text{ et } (n - |l|) \text{ pair}$$

et :

$$R_{nl}(\rho) = \sum_{s=0}^{(n-|l|)/2} \frac{(-1)^s \rho^{n-2s}(n-s)!}{s!(\frac{n+|l|}{2} - s)!(\frac{n-|l|}{2} - s)}$$

En projetant l'image $I(x, y)$ dans l'espace vectoriel défini par les polynômes, on obtient les moments complexes de Zernike avec :

$$A_{nl} = \frac{n+1}{\pi} \int\int_{x^2+y^2=1} I(x, y)V_{nl}^*(x, y)dxdy = A_{n,-l}^*$$

Il est alors intéressant de noter que l'on peut exprimer ces moments en fonction des moments centrés normalisés vus plus haut. Il faut alors utiliser la forme radiale des polynômes $R_{nl}(\rho)$:

$$R_{nl}(\rho) = \sum_{k=l}^{n} B_{nlk}\rho^k \text{ avec } (n-k) \text{ pair}$$

avec :

$$B_{nlk} = \frac{(-1)^{(n-k)/2}[(n-k)/2]!}{[(n-k)/2]![(l+k)/2]![(k-l)/2]!}$$

On a alors :

$$A_{nl} = (n+1)/\pi \sum_{k=l}^{n \sum_{j=0}^{q} \sum_{m=0}^{l} (-1)^m C_j^q C_m^l} B_{nlk}\mu_{k-2j-l+m,2j+l-m} \text{ avec } q = (k-l)/2$$

Une fois ces moments calculés, on peut en extraire des invariants aux transformations affines. Pour ce faire, on peut citer les approches de Teague [Tea79] et de Belkasim [BSA91].

1.2.3 Application d'une transformation globale

1.2.3.1 Les descripteurs de Fourier

Posons comme hypothèse de départ que le contour de l'objet est une courbe fermée de longueur L. A partir de cette courbe nous pouvons déduire une fonction $\phi(t)$ qui représente les changements de direction angulaire le long de la courbe. Ainsi on aura :

$$\phi(t) = \theta(t) - \theta(0)$$

où $\theta(t)$ représente l'angle au point $(x(t), y(t))$. On a alors :

$$\phi(0) = 0 \text{ et } \phi(L) = 0$$

Il est possible alors de normaliser cette fonction sur l'intervalle $[0, L]$:

$$\phi^*(t) = \phi(\frac{tL}{2\pi})$$

Cette fonction étant alors périodique, on peut la décomposer en série de Fourier :

$$\phi^*(t) = \sum_{k=-\infty}^{\infty} C_k e^{ikt} \text{ avec } C_k = \frac{1}{2\pi}\int_0^{2\pi} \phi^*(t)e^{ikt}dt$$

Les descripteurs de Fourier [ZR72] sont alors définis par le module des coefficients C_k.

Une autre approche consiste à considérer, non pas la variation de l'angle lors du parcours de la courbe, mais les variations propres de la courbe [Gra72]. On a alors deux fonctions à considérer : $x(t)$ et $y(t)$ avec $t \in [0, L]$. Ces fonctions étant périodiques, on peut alors aussi les décomposer en séries de Fourier :

$$x(m) = \sum_{k=-\infty}^{\infty} a(n)e^{jnw_0m} \text{ et } y(m) = \sum_{k=-\infty}^{\infty} b(n)e^{jnw_0m} \text{ où } w_0 = 2\pi/L$$

Les coefficients de Fourier sont donnés par :

$$a(n) = \frac{1}{L-1}\sum_{m=1}^{L-1} x(m)e^{-jnw_0m} \text{ et } \; b(n) = \frac{1}{L-1}\sum_{m=1}^{L-1} y(m)e^{-jnw_0m}$$

Pour être invariant à la rotation, il faut prendre :

$$r(n) = \sqrt{|a(n)|^2 + |b(n)|^2}$$

L'invariance à l'échelle s'obtient en faisant :

$$s(n) = \frac{r(n)}{r(0)}$$

Il existe aussi une autre approche, qui considère le contour comme un ensemble d'ellipses [KG82], chacune d'elles étant définie par une orientation et une position par rapport à la précédente. Les études comparatives [TOD90] montrent la supériorité de l'approche basée sur les ellipses par rapport à l'approche classique, mais cela se traduit aussi pas un coût plus important en terme de calcul.

1.2.3.2 Generic Fourier Descriptor

Les descripteurs génériques de Fourier ont été introduits par Zhang [ZL02]. Pour obtenir une invariance à la rotation, l'image subit tout d'abord une conversion à partir de coordonnées polaires (voir Fig 1.3). Puis on applique une transformée de Fourier 2D qui a finalement l'expression suivante :

$$pf(\rho, \psi) = \sum_{r=0}^{R}\sum_{i=0}^{T} f(r, \theta_i)exp(j2\pi(\frac{r}{R}\rho + \frac{2\pi i}{T}\psi)$$

avec $\theta_i = 2\pi i/T$ et R et T, les fréquences radiales et angulaires.

Les descripteurs génériques de Fourier sont alors définis par :

$$GFD = \{ \frac{|pf(0,0)|}{aire}, \frac{|pf(0,1)|}{|pf(0,0)|}, ..., \frac{|pf(0,n)|}{|pf(0,0)|}, ..., \frac{|pf(m,0)|}{|pf(0,0)|}, ..., \frac{|pf(m,n)|}{|pf(0,0)|}\}$$

où *aire* est l'aire du disque englobant l'image. m et n sont les fréquences radiales et angulaires maximales. Les auteurs proposent $m = 4$ et $n = 9$ ce qui donne en tout une signature de 36 valeurs.

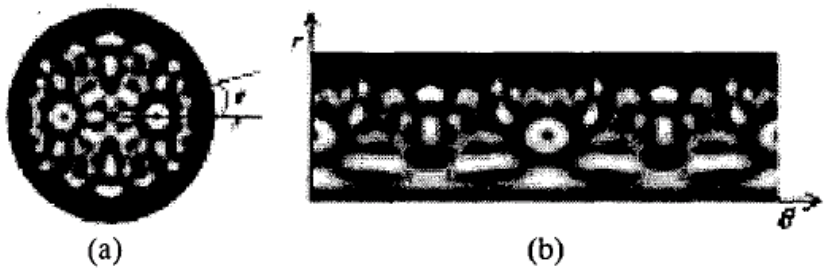


FIG. 1.3 – (a) l'image originale en coordonnées polaires ; (b) l'image transformée en coordonnées cartésiennes (repris de [ZL02]).

1.2.3.3 Descripteur basé sur la transformation de Radon

La transformée de Radon [Dea83] T_{Rf} d'une fonction f est définie par :

$$T_{Rf}(\rho,\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\delta(x\cos(\theta) + y\sin(\theta) - \rho)dxdy$$

avec :

- $-\infty < \rho < \infty$
- $0 \leq \theta < \pi$
- δ, la fonction de Dirac.

Intuitivement, ceci revient à intégrer la fonction f le long d'une droite pour n'importe quel (ρ,θ) (voir la figure 1.4).

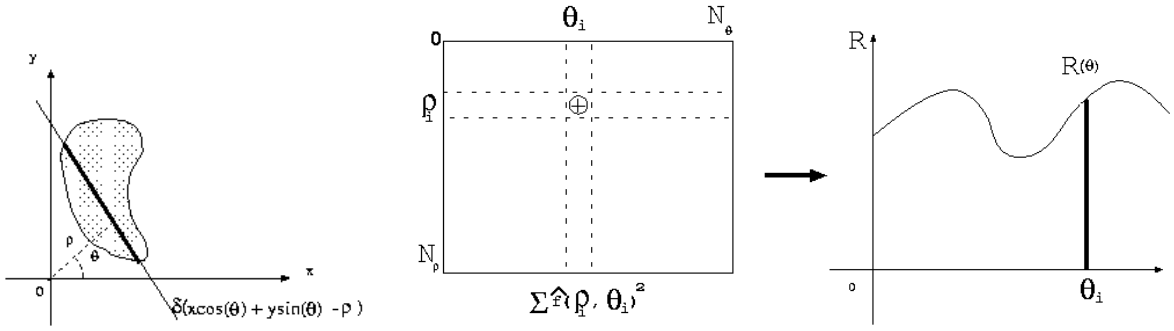


FIG. 1.4 – Définition de la transformée de Radon et calcul de la $\mathcal{R}$-signature à partir de la matrice de Radon (repris de [TW02]).

Signature issue de la transformée de Radon. On peut alors introduire une signature appellée $\mathcal{R}$-signature [TW02; TW03], qui est définie à partir de la transformée de Radon (voir la figure 1.4) :

$$\mathcal{R}_f(\theta) = \int_{-\infty}^{\infty} T_{Rf}^2(\rho,\theta)d\rho$$

La figure 1.5 montre la transformation de Radon d'un carré, ainsi que sa $\mathcal{R}$-signature.

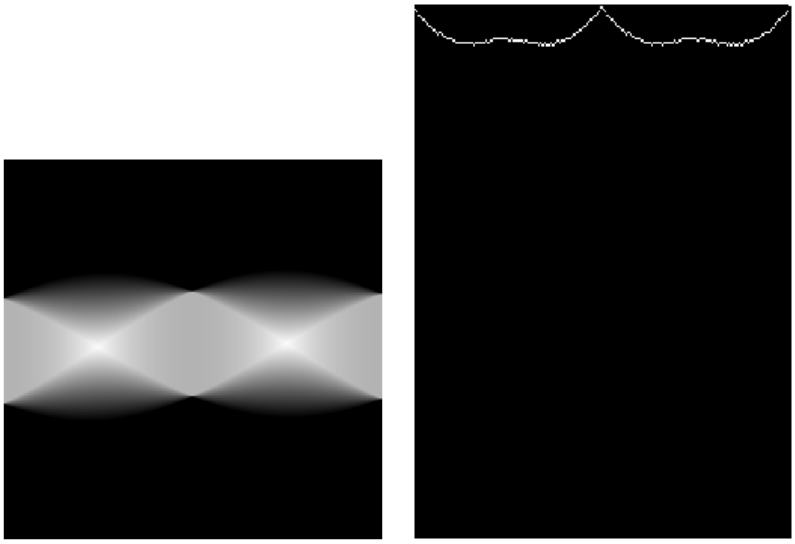


FIG. 1.5 – Transformée de Radon et signature d'un carré.

Il est facile de montrer, à partir des propriétés de la transformée de Radon, que les propriétés suivantes sont vérifiées pour la $\mathcal{R}$-signature [TW03] :

- **Périodicité de π** :
$$\mathcal{R}_f(\theta \pm \pi) = \int_{-\infty}^{\infty} T^2_{Rf}(\rho, \theta \pm \pi) d\rho = \mathcal{R}_f(\theta)$$
- **Rotation** : une rotation d'angle θ_0 sur f implique sur la $\mathcal{R}$-signature un décalage cyclique dépendant de θ_0 :
$$\mathcal{R}_f(\theta + \theta_o) = \int_{-\infty}^{\infty} T^2_{Rf}(\rho, \theta + \theta_o) d\rho$$
- **Translation** : une $\mathcal{R}$-signature est invariante à une translation de $\vec{u} = (x_0, y_0)$ sur f :
$$\int_{-\infty}^{\infty} T^2_{Rf}(\rho - x_o \cos(\theta) - y_o \sin(\theta)) d\rho = \mathcal{R}_f(\theta)$$
- **Échelle** : un agrandissement de $\alpha \neq 0$ sur f implique que :
$$\int_{-\infty}^{\infty} T^2_{Rf}(\alpha\rho, \theta) d\rho = \frac{1}{\alpha^2} \mathcal{R}_f(\theta), \alpha > 0$$

Signature 3D. Cette signature peut être employée afin de calculer une signature 3D [TWS06]. L'idée est d'utiliser une transformée de distance pour des symboles binaires. Ainsi, on obtient une image dont chaque élément représente la distance minimale au contour. L'auteur préconise l'utilisation de transformée de distance de chanfrein. L'image obtenue est ensuite segmentée en n niveaux équidistants. On calcule alors pour chaque niveau la $\mathcal{R}$-signature, ce qui nous donne une nappe 3D, que l'on utilise comme descripteur. Dans ce cas, l'auteur montre par des résultats expérimentaux que cette approche est plus discriminante.

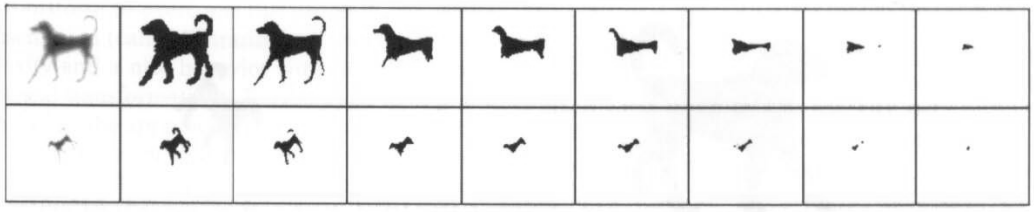


FIG. 1.6 – En première colonne : la transformée de chanfrein. Les autres colonnes : les images obtenues après segmentation des différents niveaux (repris de [TWS06]).

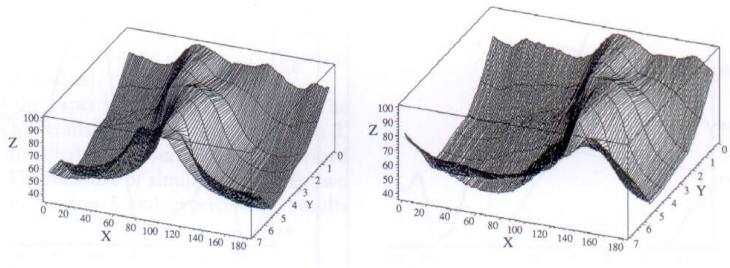


FIG. 1.7 – Les nappes 3D obtenues pour les deux symboles présentés dans la figure 1.6 (repris de [TWS06]).

1.2.3.4 Méthodes basées sur l'espace des échelles

Une autre approche proposée par Mokhatarian [MM92] utilise la notion d'espace des échelles. Le contour de l'image est échantillonné de manière régulière (l'auteur propose de ne prendre que 200 points). Le contour est ensuite représenté par sa forme paramétrique :

$$r(u) = (x(u), y(u))$$

On peut alors calculer la courbure du contour avec :

$$k(u) = \frac{x'(u)y''(u) - x''(u)y'(u)}{(x'^2(u) + y'^2(u))^{3/2}}$$

Ce qui peut se ramener à :

$$k(u) = x'(u)y''(u) - x''(u)y'(u)$$

Si la courbe est fermée, planaire, et que l'on normalise u pour que $u \in [0, 1]$

On peut lisser cette courbe en utilisant une gaussienne $g(u, \sigma)$. La courbe devient alors :

$$X(u, \sigma) = x(u) * g(u, \sigma) \text{ et } Y(u, \sigma) = y(u) * g(u, \sigma)$$

Et les dérivées successives deviennent alors :

$$X_u(u, \sigma) = x(u) * g_u(u, \sigma) \text{ et } X_{uu}(u, \sigma) = x(u) * g_{uu}(u, \sigma)$$

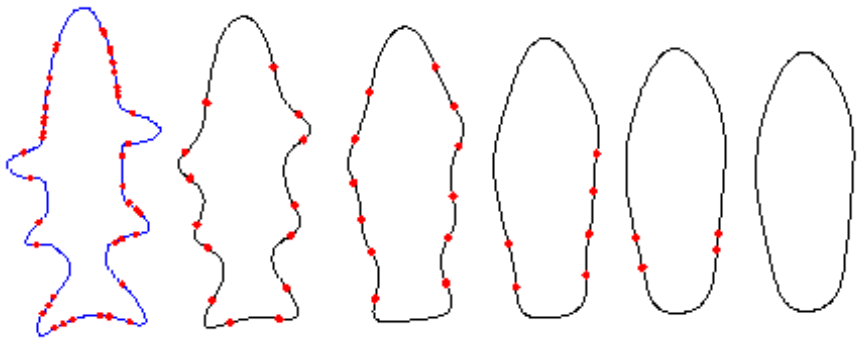


FIG. 1.8 – Les différents contours lissées, avec les points de courbures correspondants (repris de [MM92]).

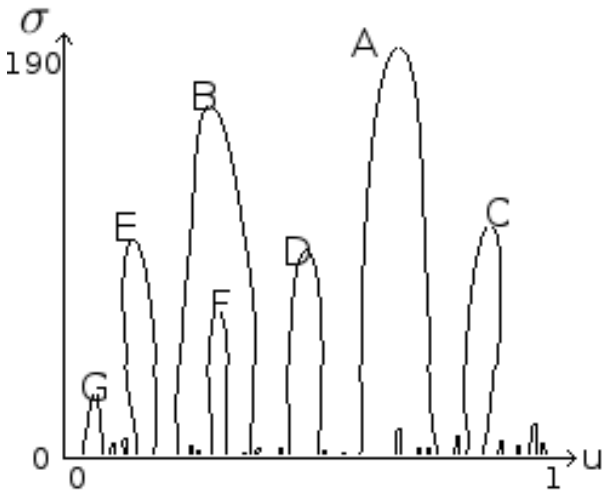


FIG. 1.9 – La position des points de courbures dans l'espace paramétrique en fonction de σ.

La courbure devient donc :

$$k(u,\sigma) = \frac{X_u(u,\sigma)Y_{uu}(u,\sigma) - X_{uu}(u,\sigma)Y_u(u,\sigma)}{(X_{uu}^2(u,\sigma) + Y_{uu}^2(u,\sigma))^{3/2}}$$

Un exemple de contour avec différents paramètres de lissage est donné en Fig 1.8.

On va ensuite noter les positions du paramètre u qui correspondent à un point de courbure (c'est-à-dire qui impliquent un changement de signe dans $k(u,\sigma)$), et ce pour différents σ que l'on incrémente de manière régulière jusqu'à ce qu'il n'y ait plus de points de courbures. On obtient au final une image semblable à celle présentée en Fig 1.9.

Le vecteur caractéristique est alors composé des maxima issus de cette image, auxquels on aura enlevé les maxima ayant un $\sigma < 0.2\sigma_{max}$. Ainsi, par rapport à la figure 1.9, on retiendra comme descripteur les coordonnées des points A à G.

L'appariement entre deux images se fait alors à l'aide d'un fonction coût qui calcule le recouvrement de deux ensembles de points en tenant compte des éventuelles permutations

circulaires. Ce descripteur a par ailleurs été retenu dans le standard MPEG-7[Bob01] en raison de son pouvoir discriminant.

1.2.3.5 Angular Radial Transform

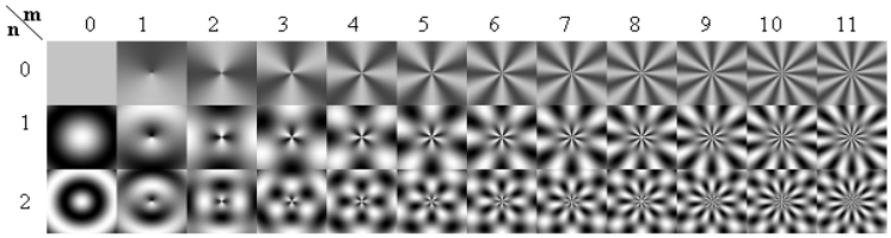


FIG. 1.10 – Partie réelle des fonctions de bases de l'ART (repris de [KK99]).

L'ART [KK99] (Angular Radial Transform) est un descripteur particulièrement efficace. En effet, il est robuste au bruit et aux changements d'échelle, invariant à la rotation, et possède une dimension raisonnable (36), ce qui lui vaut aussi d'avoir également été retenu dans le standard MPEG-7.

Ce descripteur se propose de projeter l'image sur une base orthogonale. Ainsi ses coefficients F_{nm} se calculent de la manière suivante :

$$F_{nm} = \int_0^1 \int_0^{2\pi} V_{nm}(\rho,\theta) f(\rho,\theta) d\rho d\theta$$

où $f(\rho,\theta)$ est l'image en coordonnée polaire, et V_{nm} est défini par :

$$V_{nm} = A_m(\theta) R_n(\theta)$$

$$A_m(\theta) = \frac{1}{2\pi} exp(jm\theta)$$

$$R_n(\theta) = \begin{cases} 1 & \text{si } n = 0 \\ 2cos(\pi n \rho) & \text{si } n \neq 0 \end{cases}$$

Le standard MPEG-7 préconise l'utilisation de $n = 2$ et $m = 11$, ce qui donne un vecteur de dimension 36. Le vecteur descripteur se déduit des coefficients F_{nm} avec de simples opérations. L'invariance à la rotation, s'obtient en prenant le module des coefficients. L'invariance à l'échelle s'obtient en divisant les coefficients par F_{00}. La Fig 1.10 représente la partie réelle des images sur lesquelles l'image est projetée.

1.3 Conclusion

Nous avons proposé un panorama non exhaustif de descripteurs. La principale difficulté est donc de pouvoir choisir parmi ceux-ci, les descripteurs les plus pertinents. La difficulté est d'autant plus grande que suivant les symboles recherchés, ces descripteurs seront plus ou moins performants. Dans le chapitre suivant, nous nous attarderons un peu plus sur des descripteurs qui ont attiré notre attention, puis nous les comparerons à l'aide d'une étude comparative de performance.

Chapitre 2

Etudes de descripteurs

Nous nous proposons ici d'étudier quelques descripteurs afin de pouvoir choisir ceux qui semblent pertinents pour notre application. Nous examinerons donc tour à tour la signature de Radon, l'ART, les moments de Zernike et les histogrammes de Yang. Dans un premier temps nous discuterons des paramètres des différentes méthodes. Ensuite nous nous focaliserons sur une étude comprative de performance.

2.1 La signature de Radon

Nous nous intéressons ici à la signature de Radon. Dans un premier temps, nous verrons la méthode classique pour calculer la transformée de Radon, puis nous verrons qu'il existe une méthode permettant de réduire la complexité de ce calcul. Enfin, nous mettrons en avant des améliorations qui nous permettent de nous affranchir de certains problèmes inhérents à cette signature.

2.2 Implentation

2.2.1 Calcul classique

Une manière simple d'implenter la transformée de Radon est d'utiliser l'approche proposée par [Bra95].

Soit une image I, définie par :

$$I(x,y) = \begin{cases} 1 & \text{si } x = x_O \text{ et } y = y_0 \\ 0 & \text{sinon} \end{cases}$$

On a alors :

$$I(x,y) = \delta(x - x_0)\delta(y - y_0)$$

Sa transformée de Radon est par définition :

$$T_{R^I}(\rho, \theta) = \delta(\rho - x_0 \cos(\theta) - y_0 \sin(\theta))$$

Ainsi un point a comme transformée de Radon, une courbe sinusoïdale d'équation :

$$\rho = x_0 \cos(\theta) + y_0 \sin(\theta)$$

Comme la transformée de Radon est par définition linéaire, une manière de calculer la transformée d'une image binaire consiste donc à calculer la transformée de Radon de tout ces points, puis de sommer les transformées. Pour éviter les erreurs d'approximation relatives à la théorie de Shannon , on choisit comme pas d'incrémentation (voir [Tof96] pour plus de détails) :

$$\Delta\theta = \Delta\rho = 1$$

L'algorithme est en $O(N^2M)$ pour une image de taille $N \times N$ et M angles différents (ici $M = 180$). Pour obtenir la signature, il suffit de calculer la somme carrée des éléments d'une même colonne (qui représente un thêta fixé). On obtient alors M coefficients que l'on norme sur $[0, 1]$.

2.2.2 Amélioration du descripteur

La signature est un ensemble de 180 valeurs. Le problème qui se pose est l'invariance à la rotation. En effet la signature n'est pas invariante à la rotation, mais simplement cyclique. En effet un décalage de θ_0 va induire une décalage cyclique de θ_0 dans la signature. Ainsi si nous voulons savoir si deux symboles sont identiques, il nous faut évaluer l'erreur quadratique moyenne (ou une autre mesure) sur toutes les permutations circulaires, ce qui est contraignant en terme de complexité. Une manière d'éviter cela est de calculer la transformée de Fourier rapide sur ces 180 valeurs. On ne retient alors par exemple que les 32 premiers coefficients (hormis le premier qui ne représente pas d'intérêt puiqu'il représente la somme des pixels). Ainsi le problème de la rotation est évité.

Afin de réduire la complexité, on peut calculer la transformée de Radon avec la méthode proposée par [Bra98]. L'idée consiste à calculer de manière récursive les lignes qui balayent l'image suivant un certain θ. On comprend bien qu'à partir d'un même point, la ligne traversant ce point pour $\theta = 0$ a beaucoup de points en commun avec la ligne traversant ce même point pour un θ proche de 0. La méthode utilise donc des sommes partielles afin de réduire la complexité à $O(N^2 logN)$. Cependant, cette méthode produit une transformée de Radon qui n'est pas échantillonée de manière uniforme aussi bien en θ que en ρ. Dans ce cas, la signature définit perd alors toutes ses propriétés. Des traitements sont alors nécessaires pour conserver les propriétés d'invariance aux transformations affines. Notre adaptation de la transformée rapide à une signature normalisée est décrite en annexe.

2.3 Evaluation des différents paramètres à ajuster

Pour évaluer ce descripteur avec les deux méthodes, nous avons tout d'abord testé ce descripteur sur un cercle et sur un carré de différentes dimensions (64, 128, 256 et 512 pixels). Puis nous avons calculé l'erreur quadratique moyenne entre la signature calculée avec ces deux méthodes avec la signature théorique (Fig 2.1). Il en ressort que cette erreur est négligeable dans les deux cas puisque dans le pire des cas on a moins de 0.02 d'écart (les signatures de Radon sont pour rappel normalisée entre 0 et 1).

En ce qui concerne le gain de temps, la figure 2.2 regroupe le temps de calcul nécessaire pour calculer la transformée de Radon en utilisant la méthode classique ou la méthode rapide, pour des images de tailles 128x128, 256x265, 512x512 et 1024x1024. Le gain en complexité théorique entre les deux méthodes en $N/logN$ est bel et bien confirmé.

Signature	Carre 64	Carre 128	Carre 256	Carre 512
Radon Normale	0.001	0.006	0.006	0.018
Radon Rapide	0.007	0.006	0.006	0.008
Signature	Cercle 64	Cercle 128	Cercle 256	Cercle 512
Radon Normale	<0.001	<0.001	<0.001	<0.001
Radon Rapide	0.007	0.003	0.012	<0.001

FIG. 2.1 – Ecart moyen entre la signature théorique et la signature calculée pour un cercle et pour un carre de diamètre ou de côté donné.

Signature	128x128	256x265	512x512	1024x1024
Radon Normale	1s	2s	6s	28s
Radon Rapide	<1s	1s	2s	6s

FIG. 2.2 – Temps de calcul en secondes d'une image de différentes dimensions en secondes, avec les deux méthodes de calcul.

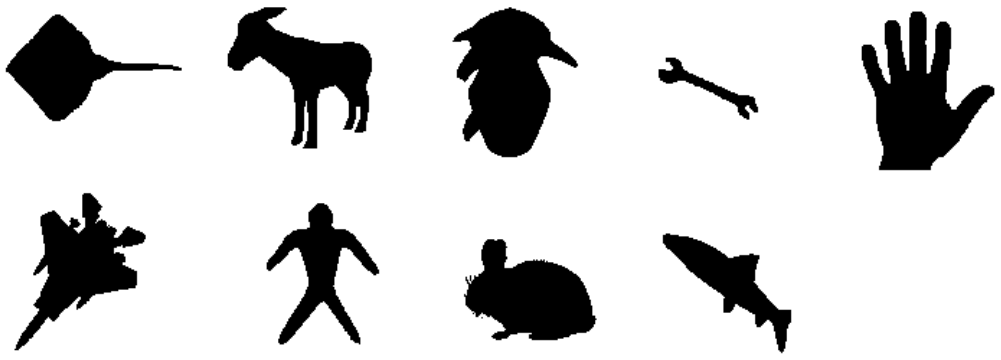


FIG. 2.3 – Symboles issus de la base B1.

Puis, nous avons évalué ce descripteur (voir Fig 2.4) sur une base notée B1 (voir Fig 2.3) de 9 symboles contenant 11 occurrences chacun, en utilisant la transformée de

Radon Rapide et de la Radon Normale, et avec et sans la transformée de Fourier. Nous avons aussi fait varier le nombre de coefficients de Fourier afin de choisir l'influence que celui-ci peut avoir.

	Nombre de coefficients de Fourier	Taux de reconnaissance	Temps de calcul
Radon Normale	0 (cad sans Fourier)	61.2 %	45s
Radon Normale	2	58.0 %	46s
Radon Normale	4	63.3 %	46s
Radon Normale	8	63.4 %	46s
Radon Normale	16	63.3 %	46s
Radon Normale	32	63.3 %	46s
Radon Rapide	0 (cad sans Fourier)	61.9 %	14s
Radon Rapide	2	58.6 %	15s
Radon Rapide	4	62.8 %	15s
Radon Rapide	8	63.3 %	15s
Radon Rapide	16	63.4 %	15s
Radon Rapide	32	63.4 %	15s

FIG. 2.4 – Taux de reconnaissance et temps de calcul en secondes sur une base de 99 images, en utilisant la méthode classique et rapide du calcul de la transformée de Radon, suivi ou non d'une transformée de Fourier.

Il ressort tout d'abord que l'utilisation de la transformée de Fourier, contrairement à ce qu'on pouvait s'attendre, améliore légèrement les résultats (de l'ordre de 2 %). Ceci peut s'expliquer par le fait que la transformée de Fourier va "lisser" la signature, et donc s'affranchir d'un éventuel bruit. Les avantages d'utiliser la transformée de Fourier sont donc multiples, cela permet :

- de réduire la taille du vecteur caractéristique (à l'origine la signature a une taille de 180)
- de réduire le coût de l'appariement (plus besoin de faire des permutations circulaires)

De plus l'utilisation de la méthode de calcul rapide de la transformée de Radon n'a que peu d'influence en terme de taux de reconnaissance. Les résultats sont, en effet, assez similaires comparés à la méthode de calcul classique. Par contre, le gain en temps de calcul est évident. Quant au nombre de coefficients de Fourier à garder, il se situe aux environs de 8, puisqu'au-delà, le gain en terme de performances est négligeable voire nul. On en déduit que pour utiliser la signature de Radon au maximum de ses capacités, il faut utiliser la méthode de calcul rapide de la transformée de Radon, suivi d'une transformée de Fourier en ne gardant que les 8 premiers coefficients (hormis le premier coefficient comme nous l'avons déjà dit).

2.4 ART

Nous avons vu au chapitre précédent que la méthode ART dépend de 2 paramètres n et m qui correspondent respectivement aux fréquences angulaires et radiales. Nous nous proposons alors de regarder l'influence de ces paramètres sur la courbe de précision/rappel en utilisant la base B1 (voir Fig 2.3). La figure 2.5 montre le gain moyen en pourcentage de la courbe précision/rappel pour les différents paramètres n et m, par rapport à la courbe de précision rappel obtenu avec n=2 et m=11. On remarque alors que sur une plage de valeur finalement assez étendue, les résultats sont stables et confirment donc l'utilisation de l'ART avec $n = 2$ et $m = 11$.

n m	8	9	10	11	12	13	14
1	0%	0%	-1%	-1%	-1%	+1%	+1%
2	+1%	+1%	0%	0%	0%	+1%	+2%
3	+2%	+2%	+1%	+1%	+1%	+2%	+3%

FIG. 2.5 – Ecart moyen en pourcentage de la courbe précision/rappel pour les différents paramètres n et m, par rapport à la courbe de précision rappel obtenue avec n=2 et m=11.

2.5 Les moments de Zernike

Comme nous l'avons noté plus haut (cf 1.2.2.2), les moments de Zernike sont construit à partir des polynômes de Zernike. Ceux-ci utilisent un paramètre n qui est appelé l'ordre. Nous nous proposons maintenant de regarder l'influence de cet ordre sur la courbe de précision/rappel toujours en utilisant la base B1 (voir Fig 2.3). La figure 2.6 montre le gain moyen en pourcentage de la courbe précision/rappel pour différentes valeurs de n, par rapport à la courbe de précision/rappel obtenue avec n=7. Les résultats montrent ici une forte influence de l'ordre puisqu'avec un ordre de 3, les performances chutent de près de 9%. Par contre avec les ordres supérieurs à 7, les performances sont quasi identiques (gain de 1% au mieux). Ces résultats confirment donc l'utilisation d'un ordre fixé à 7.

n	2	3	4	5	6	7	8	9	10
	-14%	-9%	-5%	-3%	-2%	0%	0%	+1%	+1%

FIG. 2.6 – Ecart moyen en pourcentage de la courbe précision/rappel pour les différentes valeurs de n, par rapport à la courbe de précision/rappel obtenue avec n=7.

2.6 Histogrammes de Yang

L'approche de Yang consiste à calculer deux histogrammes qui tiennent comptent de la distribution des angles entres les différents points de la formes, et de la distribution de rapport minimum de distance(cf 1.2.1.3). Il n'y a ici que les échantillonages des deux

histogrammes à fixer (ahr pour les angles, lhr pour le rapport des longueurs). La figure 2.7 montre le taux de reconnaissance obtenue sur la base B1, en faisant varier ces échantillonages. Il en ressort que pour la base utilisée, ce descripteur est finalement peu sensible au variations que l'on peut lui faire subir.

lhr ahr	5	10	20
0.05	75%	75%	73%
0.1	73%	74%	72%
0.2	72%	73%	71%

FIG. 2.7 – Taux de reconnaissance pour différents échantillonages des histogrammes de Yang

2.7 Evaluation des descripteurs

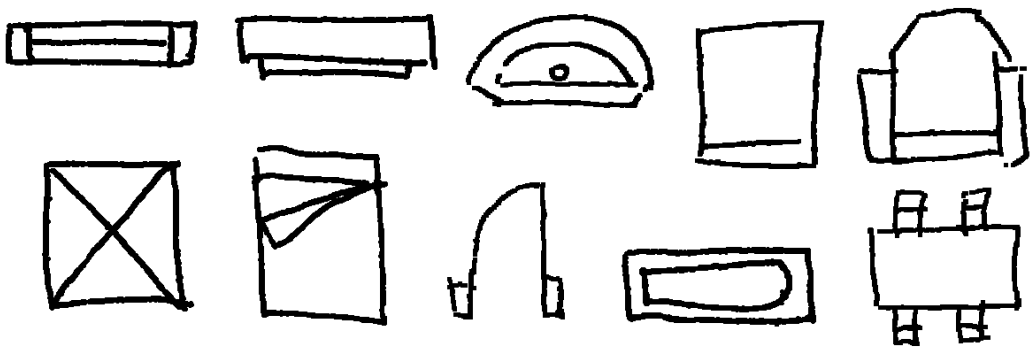


FIG. 2.8 – Symboles issus de la base B2.

Nous venons de voir les différents paramètres à ajuster afin d'utiliser au mieux la $\mathcal{R}$-signature, l'ART (voir 1.2.3.5), les moments de Zernike (voir 1.2.2.2) et les histogrammes de Yang (voir 1.2.1.3). Nous nous proposons maintenant d'évaluer ces descripteurs. Nous allons pour cela utiliser deux bases d'images. La première base B1 a déjà été présentée plus haut, la deuxième base B2 contient 3000 symboles divisés en 20 classes (voir Fig 2.8).

2.7.1 Définitions

Cette évaluation nécessite également des critères d'évaluation. Au lieu de nous contenter d'utiliser un simple taux de reconnaissance, nous allons également utiliser d'autres critères d'évaluation, comme la courbe de précision/rappel, la séparabilité [TFC04] ou encore la compactabilité [TFC04].

2.7.1.1 La courbe de précision/rappel

La précision est le ratio des images pertinentes sur l'ensemble des images renvoyées par la requête. Alors que le rappel est le ratio des images pertinentes retournées par la requête sur l'ensemble des images pertinentes contenues dans la base. On peut alors définir une courbe qui montre l'évolution de la précision en fonction du rappel. La figure 2.9 montre différentes courbes que l'on peut obtenir. La courbe idéale est celle qui correspond à une précision de 1 quelque soit le rappel. Plus les courbes se rapprochent d'une précision de 1, meilleur elles sont.

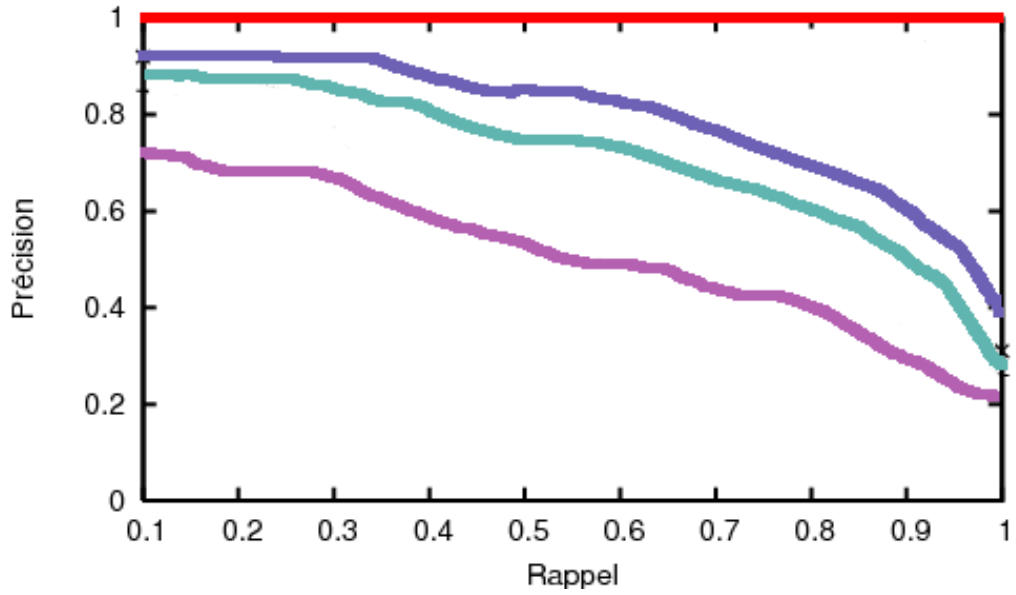


FIG. 2.9 – Différents courbes de précision/rappel

2.7.1.2 La séparabilité

La séparabilité se définie de la manière suivante. Soit un ensemble S d'objets organisés en M classes et composés de N éléments. Soit x_C un élément de la classe C contenant N_C éléments. On note $d'(x_C, i) = \frac{d(x_C,i)}{max_{j \in S} d(x_C,j)}$ où d(,) est une distance et i un élément de la base S. $d'(x_C, i)$ est alors comprise entre [0,1]. Cette valeur est alors discrétisée avec un pas de 0.02. On définit alors $\eta_{r_C}(x)$, le nombre d'objets de S qui est à une distance de r_C inférieure ou égale à x. La séparabilité $\Psi_D(C)$ d'un descripteur D pour une classe C est alors définit par :

$$\Psi_D(C) = \frac{1}{N_c} \sum_{r_C \in C} (1 - \frac{\eta_{r_C}(x)}{N})$$

La séparabilité Ψ_D du descripteur se déduit simplement avec :

$$\Psi_D(C) = \frac{\sum_{C \in S} \Psi_D(C)}{M}$$

La séparabilité est donc une courbe 2D qui rend compte du pourcentage moyen de symboles appartenant à la même classe à l'intérieur d'une hyper-sphère dont le rayon est progressivement augmenté. Ainsi, un descripteur séparable devra avoir une séparabilité proche de 1 pour des distances faibles, et proche de 0 pour des distances éloignées, la transition entre ces deux états devant se faire le plus rapidement possible pour éviter justement toute ambiguité entre les différentes classe. La figure 2.10 montre différentes courbes que l'on peut obtenir. La courbe représentant le cas idéal est celle qui est une fonction marche c'est à dire celle qui a une séparabilité de 1 ou de 0, la transition se faisant brutalement. Un descripteur séparable devra donc donner une courbe se rapprochant le plus possible de celle-ci.

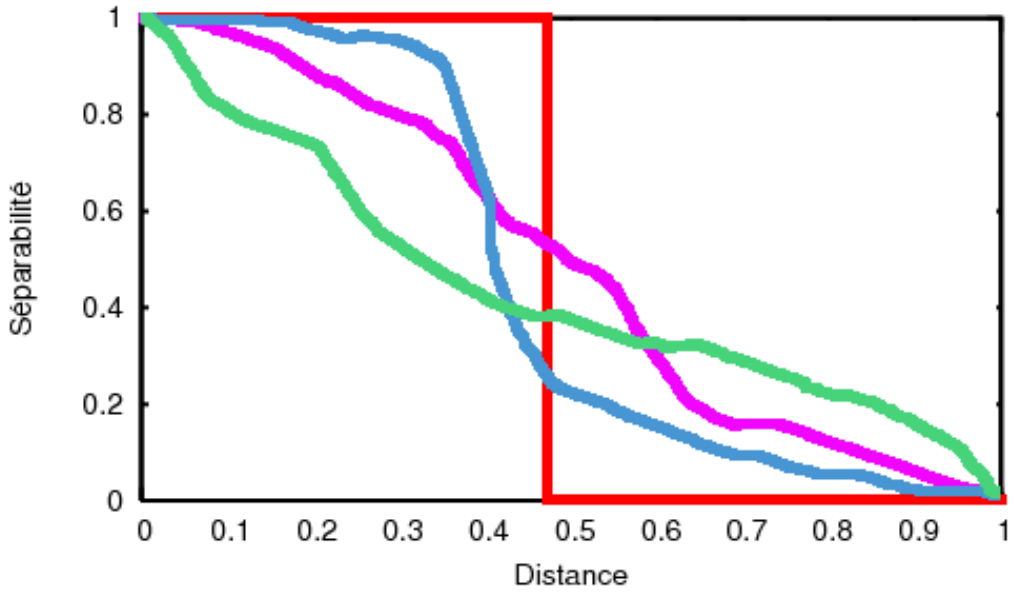


FIG. 2.10 – Différents courbes de précision/rappel

2.7.1.3 La compactabilité

La compactabilité se calcule quant à elle de la manière suivante (en prenant les mêmes notations introduites pour la séparabilité). La compactabilité $\phi_D(C)$ d'un descripteur D pour une classe C donné est définit par :

$$\phi_D(C) = \frac{1}{max_{\forall (i,j) \in S} d(i,j)} \frac{\sum_{(i,j) \in C} d(i,j)}{N_C^2}$$

La compactabilité ϕ_D d'un descripteur s'obtient alors en moyennant sur toutes les classes :

$$\phi_D = 1 - \frac{\sum_{\forall C \in S} \phi_D(C)}{M}$$

La compactabilité rend compte du fait que le descripteur arrive à former des classes qui sont le plus compact possible. Plus cette valeur est élevé, plus le descripteur à réussi à former des classes compactes.

2.7.2 Les résultats

Dans un premier temps nous donnons sur la figure 2.11 les taux de reconnaissances obtenus par les différents descripteurs sur les deux bases. L'ART et Zernike donnent des résultats proches et Yang est légérement en retrait. La signature de Radon donnent de moins bons résultats. Cependant selon Tabbone [TWS06] la signature de Radon 3D obtient dans le cas de la base B1 un score de 72%, ce qui est une nette amélioration (plus de 10%), au détriment toutefois d'une complexité encore plus élévé.

Base	Radon	ART	Zernike	Yang
B1	63%	75%	78%	74%
B2	55%	67%	65%	63%

FIG. 2.11 – Taux de reconnaissance sur la base B1 et B2 pour différents descripteurs.

Nous donnons ensuite sur les figures 2.12 et 2.13 les courbes de précision/rappel obtenus par les différents descripteurs sur les deux bases. Là encore, l'ART et Zernike se partage la première place avec Yang légèrement moins performant. La signature de Radon est encore fois en retrait, mais obtient tout de même des résultats corrects.

En ce qui concerne la séparabilité, les résultats sont donnés dans les figures 2.14 et 2.15. Les moments de Zernike sortent alors nettement du lot, puisqu'ils correspondent tout à fait à l'idée que l'on peut se faire d'un descripteur ayant une bonne séparabilité, puisque

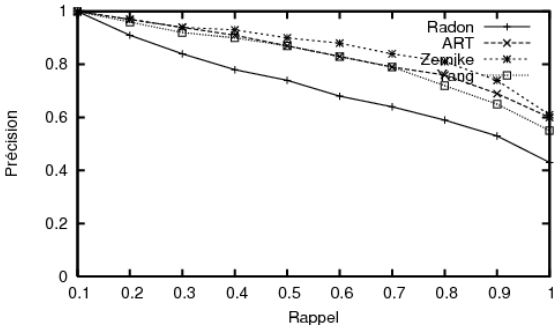


FIG. 2.12 – Courbes de rappel/précision sur la base B1 pour différents descripteurs.

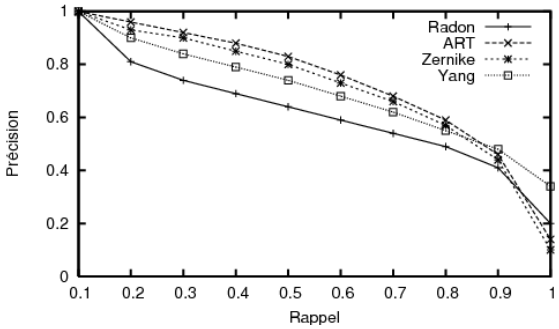


FIG. 2.13 – Courbes de rappel/précision sur la base B2 pour différents descripteurs.

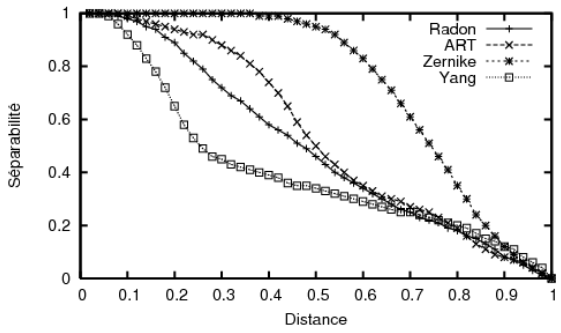


FIG. 2.14 – Courbes de séparabilité pour différents descripteurs avec la base B1.

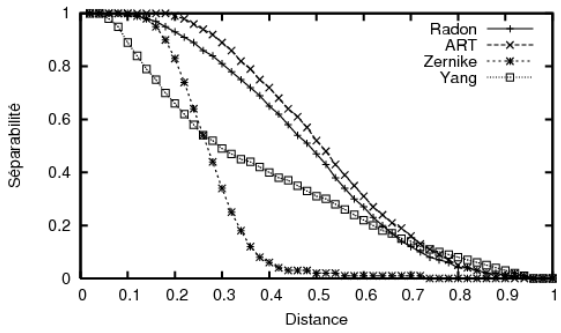


FIG. 2.15 – Courbes de séparabilité pour différents descripteurs avec la base B2.

la transition entre une séparabilité de 1 et une séparabilité de 0 se fait très rapidement (en particulier pour la base B2). L'ART et la signature de Radon obtiennent des résultats similaires et ont eux aussi une courbe de séparabilité intéressante. Par contre Yang est nettement moins performant de ce côté là.

Pour la compactabilité, les résultats sont donnés sur la figure 2.16. Il en ressort que Yang a une très forte compactabilité suivit de la signature de Radon. L'ART et Zernike ont eux par contre des compactabilités très disparates selon la base.

Base	Radon	ART	Zernike	Yang
B1	0.89	0.84	0.71	0.92
B2	0.85	0.78	0.86	0.90

FIG. 2.16 – Compactabilité sur les bases B1 et B2 pour différents descripteur.

L'ensemble de ces résultats nous montre la nécessité de choisir un descripteur selon l'utilisation que l'on compte en faire. Dans notre cas, étant davantage intéressé par avoir une bonne courbe de précision/rappel et puisque par la suite les symboles que nous sommes susceptibles de rencontrer se rapprochent plus de la base B2, il nous paraît alors judicieux de choisir l'ART.

2.8 Conclusion

Nous avons dans ce chapitre les différents paramètres à ajuster pour obtenir des résultats optimaux avec différents descripteurs. Cette étude des paramètres a ensuite été suivi par une étude de performance entre ces différents descripteurs. Il en ressort que l'ART présente dans les cas que nous avons étudiés les meilleurs performances par rapport à nos besoins. De plus la complexité le rend tout à fait utilisable dans le cadre d'une utilisation avec de grandes bases de données. C'est donc ce descripteur qu'il est préférable d'utiliser. Par la suite, c'est lui que nous utiliserons et que nous préconiserons. Dans le chapitre suivant, nous aborderons le cœur du problème qui est de pouvoir extraire des documents

des zones susceptibles de contenir des symboles. Pour cela nous mettrons en avant une méthode originale qui permet d'extraire ces zones avec un minimum de connaissances sur les symboles à extraire.

Deuxième partie

A propos de la segmentation

Chapitre 3

Segmentation des symboles

3.1 Introduction

Nous avons vu dans les chapitres précédents, les différents descriptions que l'on pouvait associer à un symbole à partir de l'image de celui-ci. Mais quand on regarde un document graphique, les symboles que l'on peut y trouver sont rarement déconnectés. En effet les symboles sont souvent connectés au graphique (comme dans les schémas électriques fig 3.1). Il s'en suit que l'on doit au préalable, d'une manière ou d'une autre, passer par un étape de segmentation. Cette segmentation permet d'isoler les symboles de leur contexte afin de pouvoir ensuite être capable de les reconnaître. Il existe donc différentes méthodes afin de réaliser cette segmentation. Nous nous proposons maintenant d'en citer quelques-unes.

3.1.1 Segmentation basée sur des traitements appliqués au document

Un document graphique est souvent très riche en information. Il est donc judicieux d'essayer d'éliminer alors ce que l'on pense ne pas être des symboles graphiques. Différents traitements existent dans la littérature, leur objectif étant de décomposer le document en différentes couches.

Couches texte et graphique. La séparation texte/graphique [FK88] permet de séparer ce que l'on considère comme du texte, de ce que l'on considère comme étant du graphique (et donc contenant les symboles). L'approche se base sur la taille des composantes connexes. Elles permettent d'extraire le texte quand celui-ci n'est pas collé au graphique.

Couches traits forts et traits fins. On peut aussi dans certains cas vouloir séparer les traits forts des traits fins [NL90] qui peuvent, avoir selon les domaines, des implications

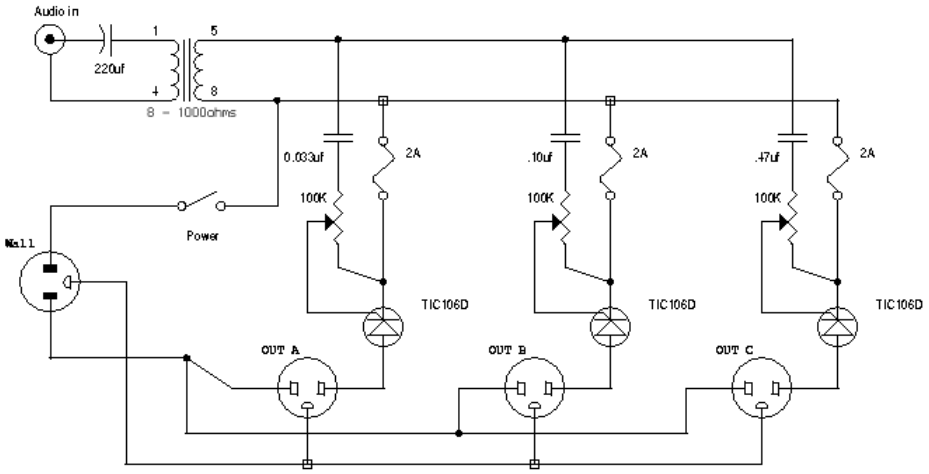


FIG. 3.1 – Schémas électriques où les symboles sont connectés au graphique.

quant à la présence ou non de symboles. Lassaulzais [LMG+98] extrait ces deux couches à l'aide d'opérations morphologiques.

Couches des hachures. On peut aussi vouloir localiser les hachures qui peuvent être un élément pertinent pour localiser des symboles. Ceci peut s'effectuer par l'étude des lignes parallèles [Ant91]. Boatto [BCB+92] présente aussi un système capable de localiser les hachures en représentant le document sous la forme d'un graphe d'adjacence des lignes. Yamada [Yam97], quant à lui, localise les hachures en remplissant les zones contenant des lignes parallèles. LLados [LSM98] localise les hachures, qui dans son cas représente les murs dans des plans architecturaux, à l'aide de la transformée de Hough.

3.1.2 Segmentation basée sur les boucles

De nombreux symboles sont basés sur des boucles. Ainsi de nombreuses méthodes essayant d'extraire les boucles d'un document ont vu le jour. Okazaki [OKM+88] et Habacha [Hab93] analysent des plans électroniques par les boucles de composantes connexes blanches. Pour d'autres, cette recherche se fait à partir d'une vectorisation. Ainsi Yoneda [YKTF94] parcourt les segments en essayant de former des polygones à l'aide de la méthode de Kasturi [KRCO90]. Yu [YSS97] quant à lui, utilise la méthode de *Moebius*, qui consiste à ordonner dans le sens trigonométriques les segments connectés en chaque point de jonction et à parcourir ces segments en suivant le sens trigonométrique. Ces méthodes sont par définition restreintes au cas où les symboles sont composés de boucles. Ceci n'étant pas toujours le cas, d'autres types de segmentation doivent être appliqués.

3.1.3 Autres types de segmentations

Certaines approches se basent sur des hypothèses a priori plus ou moins fortes quant aux symboles à retrouver. Ainsi den Hartog [dHtKG94] fixe une taille maximale pour la taille des composantes connexes dans le document. Kasturi [KRCO90] se base sur le rectangle englobant des composantes connexes pour en fixer lui aussi une taille maximale. Chen [CLW+96] ne garde que les zones ayant une densité de pixels assez élevée. Ces approches restent cependant à notre avis très ad hoc.

3.1.4 Segmentation structurelles

La segmentation dans le cas des approches structurelles est à différencier de la segmentation que l'on a pu voir jusqu'alors. L'utilisation de graphes pour représenter le document permet de s'affranchir de la segmentation. En effet, quand un modèle est présenté, on va chercher ce graphe modèle dans le graphe représentant le document, on a alors affaire à un problème de recherche d'isomorphisme de sous-graphes. Quand on a donc trouvé un sous-graphe correspond au graphe modèle, il est forcément détaché de son environnement. Ainsi, dans le cas des graphes, segmentation et reconnaissance s'effectuent ensemble.

Il existe plusieurs méthodes d'appariements de graphes. Celles qui recherchent un isomorphisme exact vont à l'aide d'un algorithme de *backtracking* [Ber73] tester successivement tous les appariements possibles. Ullman [Ull76] tente de réduire la complexité en utilisant un algorithme de *forward checking* qui permet de rejeter dès le départ les cas qui ne peuvent satisfaire la solution. Ces méthodes d'appariements ont une complexité élevée, et ne permettent pas de prendre en compte un éventuel bruit qu'aurait pu subir les symboles. C'est pourquoi on préfère utiliser les méthodes qui recherchent un isomorphisme inexact. On tolère une marge d'erreur en échange d'un coût. Il s'agit alors de trouver les sous-graphes qui minimisent ce coût [LLKM97]. Dans le même ordre d'idée, la relaxation discrète permet de rechercher des isomorphismes de sous-graphes en éliminant des hypothèses d'appariements localement incohérentes [MH86]. Ces approches sont très intéressantes car elles ont un pouvoir de représentation élevé mais elles sont très sensibles aux erreurs de segmentation du document tributaires des traitements de bas niveaux (binarisation, vectorisation).

3.2 Présentation de notre méthode

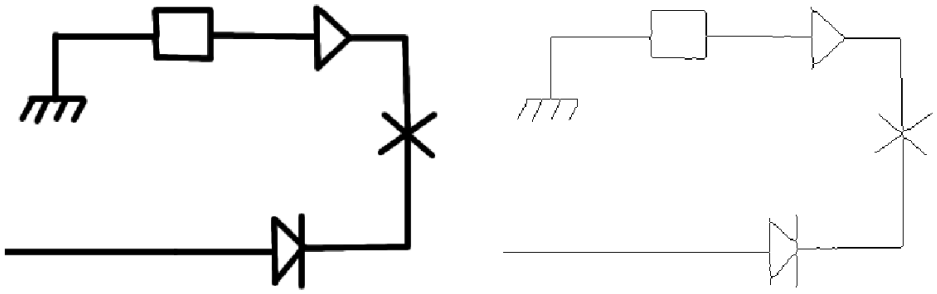
Nous proposons maintenant une méthode capable de localiser et de reconnaître des symboles dans un ensemble de documents graphiques sans avoir de connaissance a priori sur les symboles recherchés.

Notre système est composé de deux parties. La première est basée sur une méthode structurelle qui va s'aider d'une décomposition du document en chaînes de points pour générer un certain nombre de symboles candidats. La deuxième partie est composée d'un système de requête qui permet à l'utilisateur de retrouver les symboles qui l'intéressent

parmi les symboles candidats qui ont été segmentés lors de la phase de décomposition et ceci en se basant sur une description pixel des symboles.

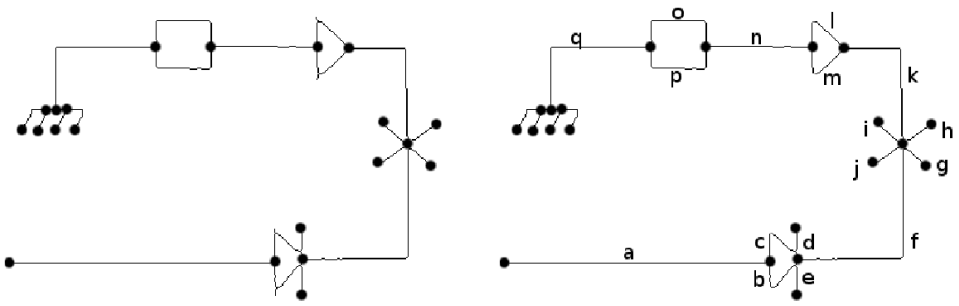
Nous présenterons en premier la méthode qui permet de détecter des symboles dans des documents graphiques. Ensuite nous décrirons ce que nous avons utilisé pour reconnaître les symboles par rapport à la requête de l'utilisateur et proposerons une évaluation expérimentale de notre méthode.

3.2.1 Présentation générale



(a) Document binarisé (b) Document squelettisé

FIG. 3.2 – Pré-traitements du document (1ère et 2ème étape).



(a) Localisation des points de jonctions (b) Décomposition en chaînes de points

FIG. 3.3 – Pré-traitements du document (3ème et 4ème étape).

Notre méthode pour localiser les symboles est basée sur des chaînes de points extraites du document. Le but est d'isoler des sous-ensembles de chaînes de points qui pourraient représenter un symbole (§3.2.2).

Une approche simple pourrait être de considérer toutes les combinaisons possibles de chaînes. Cette approche est bien entendu inapplicable dans le cadre de grands documents. En effet, ceci est dû non seulement à l'explosion combinatoire qui en résulterait, mais

aussi au fait que le grand nombre d'hypothèses qui en découlerait viendrait détériorer les performances en terme de reconnaissance.

Notre idée est d'alors de fusionner de manière itérative ces chaînes de points dans le but de reconstruire les symboles du document. Ce processus de fusion peut être représenté par un dendrogramme (§3.2.3).

3.2.2 Création d'un graphe de jonction

Pour commencer, nous décrivons le document graphique sous la forme d'un graphe de jonction. Au préalable, le document doit subir des pré-traitements préliminaires. Il doit être binarisé (Fig 3.2(a)), puis squelettisé (Fig 3.2(b)).

Une fois qu'il est squelettisé, nous allons repérer dans le squelette les points de jonctions (c'est-à-dire les points ayant au moins trois voisins), et les points terminaux (les points n'ayant qu'un seul voisin), ce qui est représenté en figure 3.3(a). La squelettisation peut malheureusement introduire de nombreux artefacts. Ainsi on trouve en pratique plus de points de jonctions que l'on voudrait, mais ce n'est heureusement pas préjudiciable pour la méthode que nous présentons ici. En effet, les noeuds indésirables ne gêneront pas la reconstruction du symbole, mais augmenteront la complexité de la méthode.

Une fois que les points de jonctions et que les points terminaux ont été trouvés, nous pouvons alors retrouver les chaînes de points connexes qui composent le document (voir Figure 3.3(b)). Ces chaînes de points sont définies par un ensemble de points connectés ne possédant que deux voisins, et dont les extrémités sont soient des points terminaux, soient des points de jonctions.

Nous construisons ensuite un graphe de jonction où les nœuds représentent les chaînes de points et où deux chaînes de points connectées sont reliées par un arc. La figure 3.4 nous montre une partie du graphe de jonction, correspondant au document présenté à la figure 3.3(b).

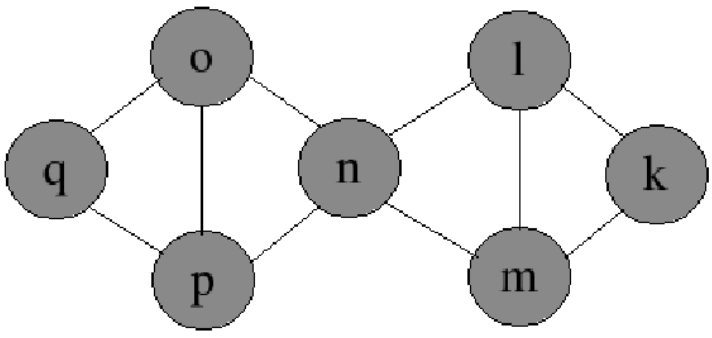


Fig. 3.4 – Une partie du graphe de jonction définie sur la Fig 3.2(b).

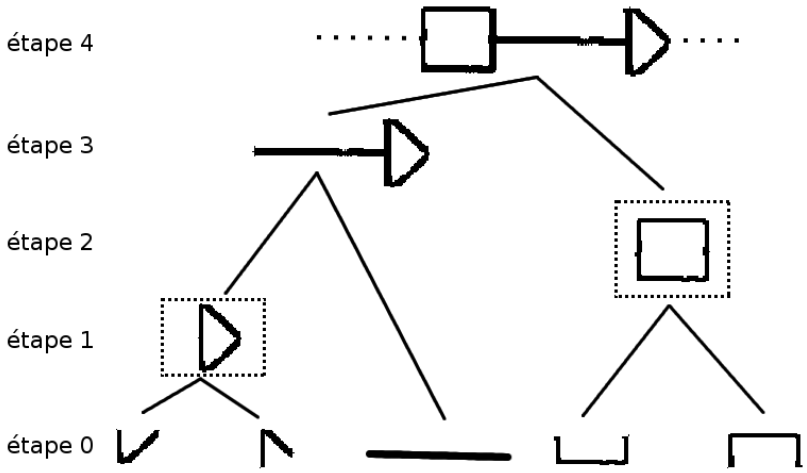


FIG. 3.5 – Un dendrogramme.

3.2.3 Construction du dendrogramme

Une fois le graphe construit, il s'agit de fusionner successivement les nœuds de ce graphe de jonction pour construire un dendrogramme. La figure 3.5 montre une partie du dendrogramme pour un document composé de deux symboles simples (un carré et un triangle) reliés par une ligne. Les nœuds entourés par un rectangle (en pointillé sur la figure) sont ceux qui correspondent aux symboles correctement reconstruits.

3.2.4 Le critère d'agrégation

Pour guider la fusion, nous avons besoin d'un critère d'agrégation qui va nous permettre de décider quelles chaînes de points doivent être fusionnées à chaque étape. Il faut alors comprendre que chaque étape représentera un symbole potentiel. Bien définir ce critère est primordial, car il faut être capable de fusionner en premier les chaînes de points appartenant aux symboles.

Pour définir un critère d'agrégation efficace, nous allons émettre les deux hypothèses suivantes quant à la nature des symboles :

- un symbole est un ensemble compact de chaînes de points connectées.
- les chaînes de points d'un symbole ont tendance à être convexes.

A partir de ces deux hypothèses, nous allons introduire deux mesures pour des chaînes de points. Une mesure sur la compacité qui sera notée m_c, et une mesure sur le degré de convexité qui sera notée m_f.

La compacité Soit un ensemble de chaîne de points C, x_C^G le barycentre de tous les points composant C, n_C le nombre de points dans C, x_i le i-ème point C, et $d(x, y)$ la distance euclidienne entre les points x et y. La compacité $m_c(C)$ de C est donnée par :

(a) la chaîne de point (b) l'aire considérée

FIG. 3.6 – Calcul du degré de convexité.

$$m_c(C) = \frac{\sum_{i=1}^{n_C} d(x_i, x_C^G)}{n_C} \tag{3.1}$$

Le degré de convexité Pour mesurer le degré de convexité $m_f(C)$ d'un ensemble de chaîne de point C, nous calculons l'aire comprise par ces chaînes de points en suivant une direction verticale et horizontale. Un exemple de l'aire calculée est donné par la figure 3.6. $m_f(C)$ de C est alors défini par le rapport entre l'aire trouvée et l'aire du rectangle englobant C.

$$m_f(C) = \frac{\text{aire trouvée(C)}}{\text{aire rectangle englobant(C)}} \tag{3.2}$$

$m_f(C)$ est à valeur dans $[0, 1]$.

Combinaisons des mesures Nous pouvons alors combiner ces deux mesures en une seule mesure qui deviendra notre critère d'agrégation :

$$m(C) = m_c(C)(1 - m_f(C)) \tag{3.3}$$

Ainsi pour un ensemble de chaînes de points C, plus ces chaînes de points seront compactes et plus $m_c(C)$ sera faible. De même, plus ces chaînes de points auront tendance à être convexes, plus $m_f(C)$ sera élevé, et donc plus $m(C)$ sera faible. Pour résumer, quand $m(C)$ est faible cela correspond aux hypothèses que nous avons introduites sur la caractérisation des symboles. On va donc utiliser cette mesure dans la construction du dendrogramme.

3.2.5 Algorithme

Nous présentons ci après, l'algorithme général pour la construction du dendrogramme.

Construire le graphe de jonctions à partir du document squelettisé.
Pour chaque chaîne de points C_i dans le graphe **faire**
  Calculer $m(C_i)$
Fin Pour
Pour chaque arc $e(C_i, C_j)$ dans le graphe **faire**
  Calculer le $m(C_i \cup C_j)$ correspondant
  Placer $e(C_i, C_j)$ dans une liste Q triée par $m(C_i \cup C_j)$ croissant
Fin Pour
Tant que (Q n'est pas vide) **faire**
  Retirer le premier élément $e(C_i, C_j)$ de la liste Q
  Créer un nouveau noeud qui sera associé à $C_i \cup C_j$
  Mettre à jour les différents arcs du graphe
  Retrier la liste Q par $m(C_i \cup C_j)$ croissant
Fait

3.2.6 Améliorations de la segmentation

La mesure introduite peut néanmoins être mise en défaut dans des documents contenant des symboles fortement connectés. Ceci est dû à une vue trop locale dans la construction du dendrogramme, qui nous empêche d'agréger des chaînes de points qui auraient pourtant donné une fusion plus intéressante plus loin dans le dendrogramme. Pour améliorer ceci, nous avons introduit une méthode de prédiction, qui au lieu de se baser simplement sur les chaînes de points qui vont effectivement fusionner, se base également sur les conséquences que cette fusion peut avoir dans les étapes suivantes. En d'autres termes, nous allons considérer non seulement les chaînes de points courantes mais aussi les chaînes de points qui y sont connectées avec une profondeur maximum n_{max} donnée.

Ainsi pour chaque $n \in [0..n_{max}]$, nous calculons le critère m pour chaque ensemble de chaînes de points. Soit m^i_{min} la mesure minimum à la profondeur i. Nous combinons alors ces minima en une seule mesure :

$$m^{n_{max}}(C_1, C_2) = \sum_{k=0}^{n_{max}} \alpha(k) m^k_{min}(C_1, C_2) \tag{3.4}$$

où α est un facteur d'atténuation.

La figure 3.7 montre l'impact de cette mesure. La figure 3.7(a) représente le symbole original dans le document, et la figure 3.7(b) est la fusion résultante que l'on trouve en gardant une vue locale. La fusion résultante que l'on trouve avec une vue plus globale de l'approche est présentée en Fig 3.7(c). Dans cette figure on a un document à segmenter (a), et un symbole à retrouver. Nous avons vu que les chaînes de points sont successivement fusionnées. En particulier, à l'étape que nous avons noté comme courante, il y a déjà eu une

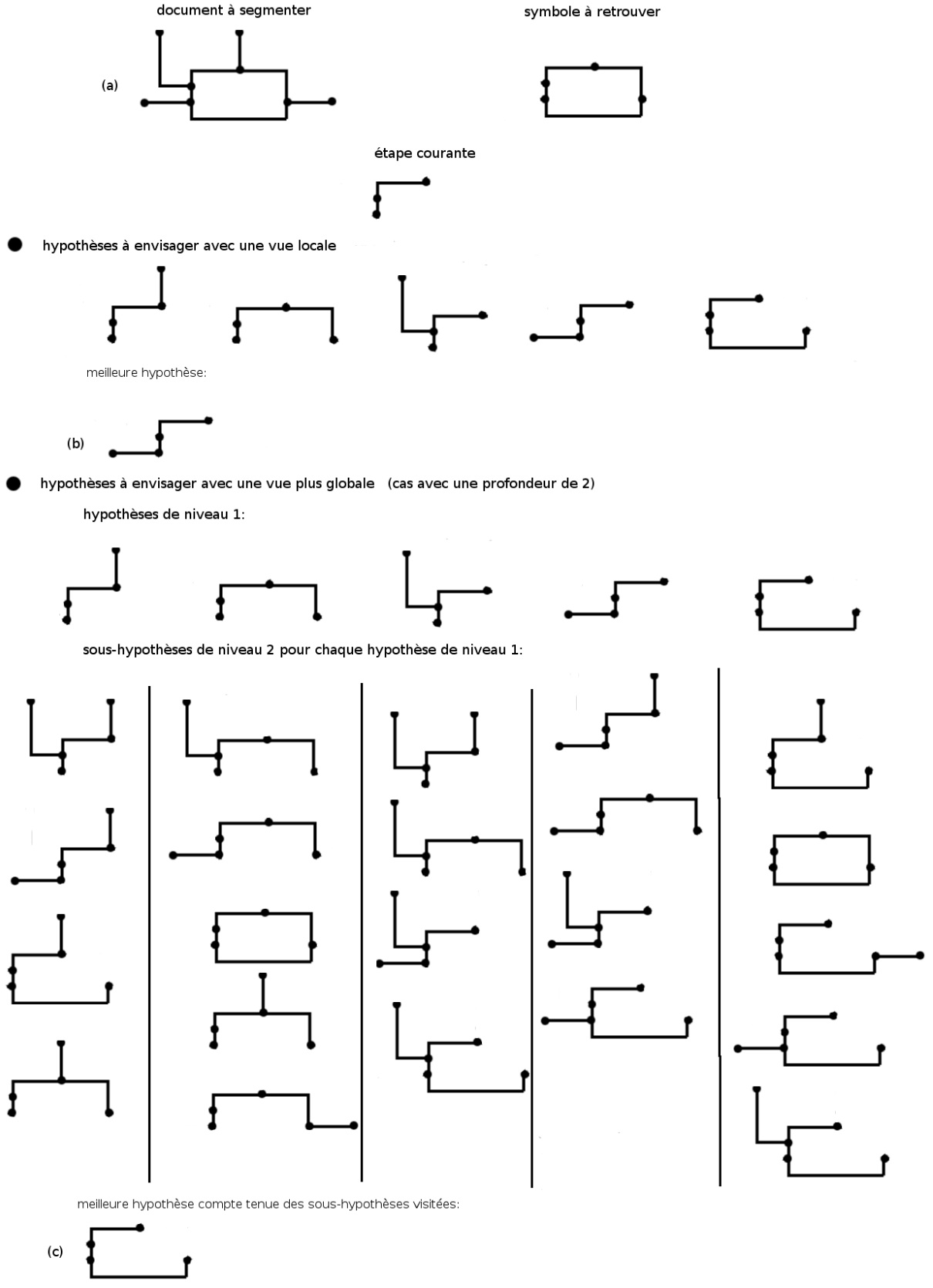


FIG. 3.7 – Symbole trouvé avec un vue locale ou globale

fusion. La question est maintenant de savoir quelle autre chaîne de point va fusionner. Avec une vue locale, on ne regarde que les chaînes de points qui sont directement connectés, ce qui nous donne cinq cas possibles. Dans ce cas, le critère que nous avons définit nous dit de fusionner de telle manière à avoir l'ensemble (b), car c'est celui qui a la compacité la plus faible. Malheureusement cet ensemble ne nous permettra pas de retrouver notre symbole. Si par contre, nous prenons une vue plus globale, on ne se limite plus seulement aux chaînes de points directement connectés (les hypothèses de niveau 1), mais on regarde plus en profondeur (les hypothèses de niveau 2). Dans ce cas, compte tenue des différentes hypothèses visitées, on obtient le symbole (c), qui va pouvoir permettre une segmentation correcte du symbole à retrouver.

Temps de calcul Les temps de calcul pour notre méthode, quand elle est utilisée dans le cas classique, reste raisonnable (de l'ordre de 20s pour des documents de 3000x3000 pixels). Par contre, ils peuvent très vite augmenter quand nous utilisons l'amélioration que nous avons proposée afin de prédire au mieux les fusions les plus prometteuses. Dans ces cas là, il est judicieux d'avoir un graphe le moins dense possible afin d'éviter l'explosion des combinaisons à explorer. Nous avons par exemple choisi de traiter les documents avec une séparation texte/graphique, afin d'éliminer au mieux les composantes inutiles du document. Si cela est déjà un premier pas, le vrai problème surgit quand nous avons dans le document des zones qui contiennent des éléments très connectés entre eux, comme un quadrillage (qui est souvent présent dans les plans d'architectures pour représenter du carrelage). Dans ces cas-là, il n'y a pas d'autres moyens que de pré-traiter le document afin d'éliminer ces zones.

3.3 Evaluation

3.3.1 La méthode d'expérimentation

La méthode a été testée sur une base d'images contenant 100 documents de schémas électriques d'avion. La taille moyenne de ces documents est de 3000x3000 pixels. La figure 3.8 montre un exemple de ces documents, tandis que la figure 3.9 montre les symboles extraits à différentes étapes données. Il faut noter qu'une étape de séparation texte/graphique a été appliquée afin de réduire la complexité de la méthode. A partir de ces documents une vérité terrain a été construite, rassemblant 1168 symboles en 18 classes différentes. Pour construire la vérité terrain, appliquer et évaluer notre méthode, nous avons dû développer une interface qui est décrite en annexe.

3.3.2 Résultats et conclusion

La méthode a été testée sur l'ensemble de ces documents en utilisant une profondeur maximale de 1 (le cas normal) et de 2 (qui sont notés S1 et S2 respectivement dans les figures suivantes). Le temps de calcul nécessaire fut respectivement de 30 minutes et de

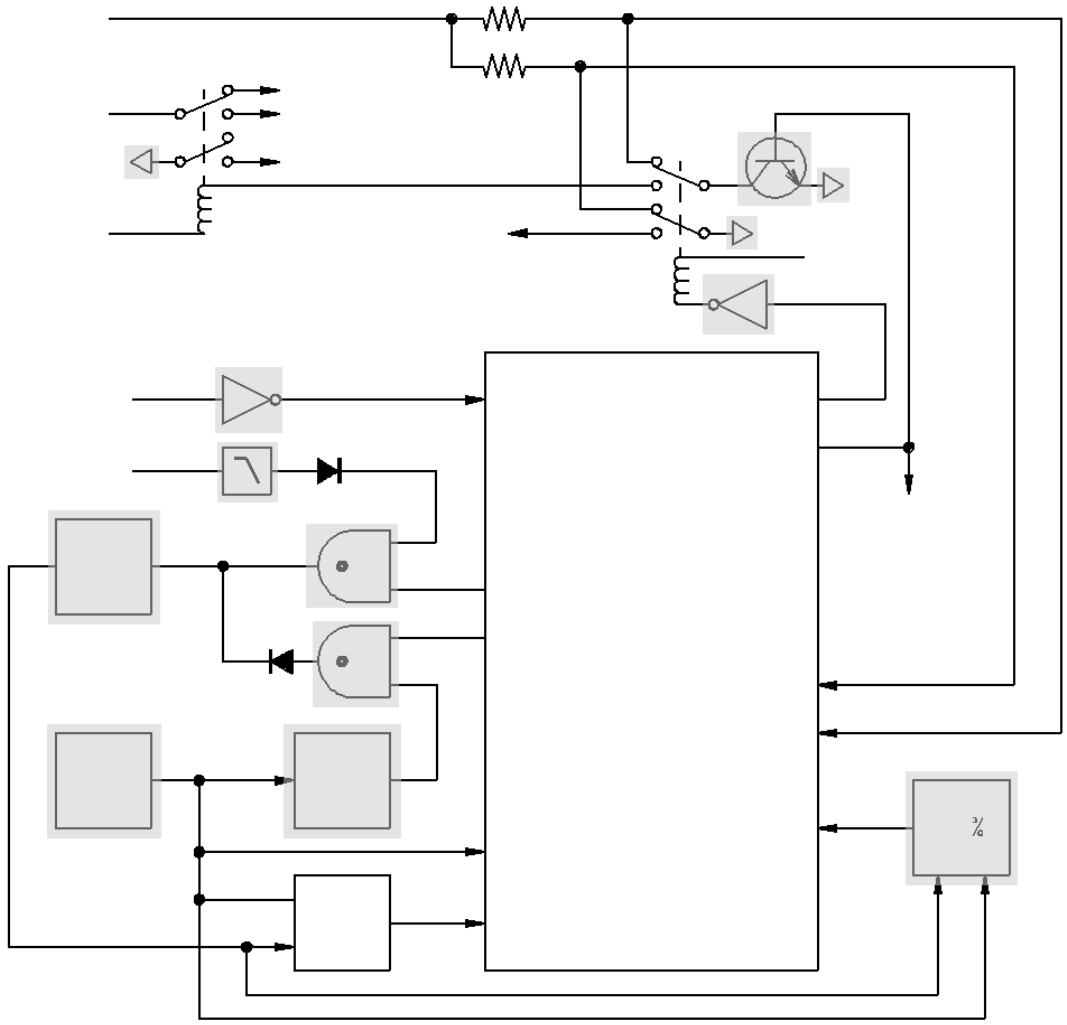


FIG. 3.8 – Exemple de document traité. Les symboles correctement segmentés sont grisés.

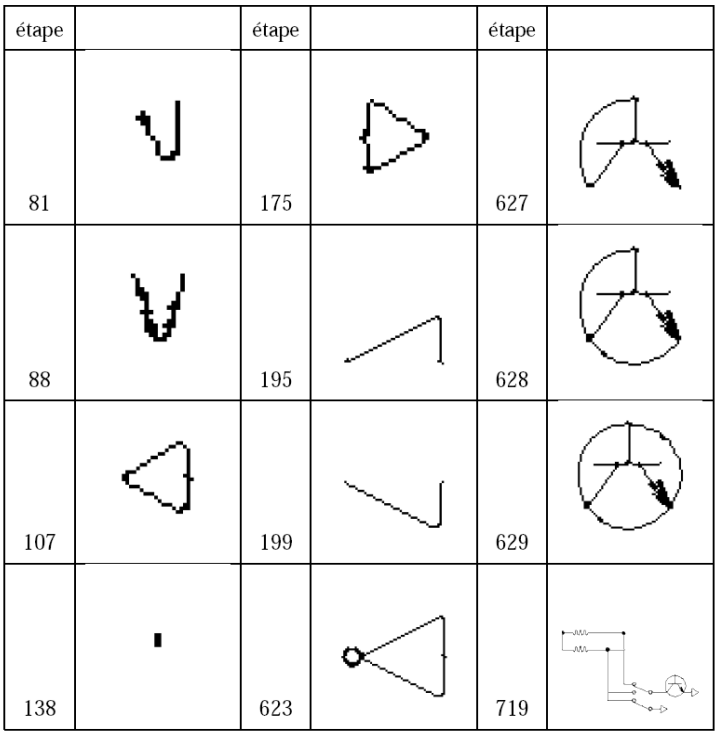


FIG. 3.9 – Exemples de reconstruction du document figure 3.8 à plusieurs étapes données.

1h20 pour la profondeur de 1 et de 2. Un vecteur caractéristique a ensuite été calculé pour chaque symbole potentiel (environ 200000 pour les 100 documents traités) en utilisant l'ART, ce qui a demandé un temps de calcul de 3h30.

Un représentant pour chaque classe a ensuite été choisi et nous avons recherché les 500 plus proches voisins. La Fig 3.10 donne les courbes de rappel/précision en utilisant l'ART avec une profondeur de 1 et de 2. Dans le cas d'une profondeur de 2, les résultats sont meilleurs, conformément à ce que l'on pourrait s'attendre. Par contre, en ne tenant compte à chaque fois que des 500 premiers plus proches voisins, on a manqué environ 660 symboles dans les deux cas, soit à peu près 57% des symboles. Ces symboles manqués peuvent être de deux natures. Soit la méthode n'a pas été à même de segmenter correctement les symboles, soit ces symboles correctement segmentés sont plus loin dans la classification (en raison d'un manque de performance du descripteur). Les résultats ne sont donc pas spectaculaires mais restent encourageant.

La deuxième donnée à prendre en compte est le temps nécessaire qu'il a fallut pour trouver les 100 plus proches voisins des 18 symboles qui est de 18s. Nous sommes alors confrontés à un double problème. Non seulement notre méthode de segmentation génère beaucoup de symboles candidats (200000 pour 100 documents traités), mais en plus nous devons utiliser un descripteur qui a une dimension relativement grande (l'ART a une dimension de 36) afin d'en tirer des résultats intéressants. Le problème est donc ici de trouver une méthode qui recherche les plus proches voisins, dans le cadre d'une base de grande taille avec des vecteurs de grandes dimensions. Nous avons jusqu'ici utilisé

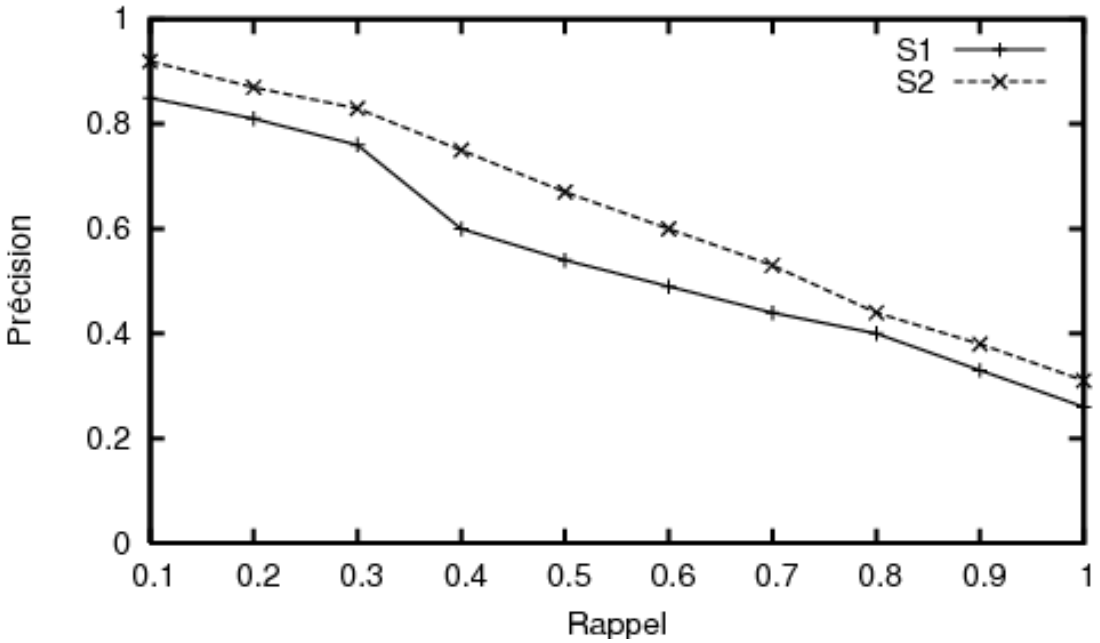


FIG. 3.10 – Rappel Précision en utilisant l'ART pour une prondeur de 1 et de 2.

une méthode linéaire, qui parcourt tous les éléments un par un. Il va de soit que cette méthode n'est pas du tout efficace dans le cadre de grande base de données ayant de plus des dimensions élevées. Nous nous proposons donc dans le prochain chapitre de regarder les différentes méthodes d'indexation qui permettront de retrouver de manière plus efficace les plus proches voisins par rapport à une requête donnée.

Troisième partie

A propos de la recherche rapide des plus proches voisins

Chapitre 4

Recherche des plus proches voisins

4.1 Introduction

Nous avons vu dans la partie précédente une méthode permettant d'extraire d'un document des zones susceptibles de contenir des symboles. Il est question maintenant de retrouver les plus proches voisins à partir d'une requête graphique. Afin que ce système soit exploitable par un utilisateur, les temps de réponse doivent être rapides. Cependant, dans notre cas, les nombreux symboles candidats générés ainsi que la taille du descripteur que nous utilisons vont ralentir fortement les temps de réponse. Il nous a donc semblé judicieux de nous attarder sur les méthodes permettant d'organiser les données, afin de réduire les temps de réponse à une requête donnée. Avant donc de proposer une méthode originale d'indexation, nous allons voir dans ce chapitre les enjeux de l'indexation multidimensionnelle, puis nous passerons en revue les différentes méthodes de *clustering* qui jouent un rôle important dans le domaine de l'indexation.

4.2 Indexation multidimensionnelle

Les techniques traditionnelles d'indexations multidimensionnelles reposent sur le principe qui consiste à regrouper en paquets les vecteurs de la base, puis à les englober dans des cellules ayant une forme simple à manipuler. L'idée est de considérer les données par paquets plutôt que de les considérer vecteur par vecteur. Lors d'une recherche, on ne travaillera que sur les paquets afin de ne sélectionner que les plus pertinents. Puis, dans un second temps, on accèdera aux vecteurs appartenant aux paquets sélectionnés. Ainsi, cela permettra de réduire le nombre d'entrées/sorties et le nombre de calculs de distance à effectuer, et donc cela diminuera de manière conséquente le temps global de recherche.

4.2.1 Formation des paquets

Il existe principalement deux stratégies de création des cellules. L'une est basée sur le partitionnement des vecteurs et l'autre est basée sur le partitionnement de l'espace.

Les algorithmes utilisant la première stratégie partitionnent les vecteurs de la base en paquets selon la distribution des vecteurs et leur proximité relative dans l'espace. A chaque paquet est associé une cellule qui englobe l'ensemble de ses vecteurs. Cette catégorie regroupe des techniques comme le **R-Tree** [Gut84], le R^+**-tree** [SRF87], le R^***-Tree** [BKS90], le **X-Tree** [BKK96], le **SS-Tree** [WJ96], le **SR-Tree** [KS97], le **TV-Tree** [LJF94], le **M-Tree** [CPZ97], ...

Les algorithmes utilisant le partitionnement de l'espace, divisent l'espace multidimensionnel directement en cellules suivant une grille plus ou moins complexe et régulière. On peut citer le **K-D-B-Tree** [Rob81] ou encore le LSD^h**-Tree** [Hen98].

Si aucun des ces algorithmes ne sort réellement du lot, des travaux ont par contre montré que si ces algorithmes effectuent des recherches de plus proches voisins de manière efficace dans des espaces de petite dimensions (<16 selon [RSB98]), leurs performances se dégradent très rapidement quand la dimension des données croît, de sorte qu'ils deviennent même moins performants qu'une recherche séquentielle et exhaustive.

4.2.2 Indexation et malédiction de la dimension

La notion de "malédiction de la dimension" a été utilisée la première fois par Bellman [Bel61]. Nous allons voir que cette expression repose sur des bases théoriques bien concrètes. Nous citerons deux propriétés des espaces à grandes dimensions qui sont tout particulièrement pénalisantes dans notre cas, pour de amples détails on pourra se référer à l'état de l'art fait par Berrani[Ber04].

Partitionnement exponentiel de l'espace. Le nombre de cellules résultant d'un partitionnement de l'espace croît exponentiellement avec le nombre de dimensions. Ainsi, dans le cas où chaque dimension d'un espace de dimension d est fractionnée en 2, le nombre de cellules formées dans cet espace est de 2^d. Ainsi, découper en deux un espace à 36 dimensions, comme dans le cas du descripteur ART, reviendrait à créer 68 milliards de cellules ! Et ce, uniquement pour découper chaque dimension de l'espace en deux !

Frontières. A l'issue d'un partitionnement, chaque cellule possède un très grand nombre de cellules voisines. Cela pose de sérieux problèmes pour les procédures de recherche. En effet, dans le cas où un vecteur requête se trouve suffisamment proche d'une frontière entre les cellules, le processus de recherche de plus proches voisins doit nécessairement examiner la ou les cellules voisines. Or, en grandes dimensions, le nombre de cellules augmente exponentiellement et la probabilité d'être proche d'une frontière augmente elle aussi. Ainsi, dans un espace à grande dimension, même s'il est possible d'éliminer de très nombreuses cellules, la proximité des frontières entraîne la quasi certitude de devoir

accèder aux cellules voisines, qui cumulées à l'existence de très nombreux voisins, tend à augmenter fortement le coût des recherches.

4.2.3 Nouvelles approches tenant compte de la dimension élevée

Afin de pouvoir gérer le cas des grandes dimensions, plusieurs méthodes ont vu le jour. Nous pouvons citer le Pyramid-Tree [BBK98] qui permet de partitionner l'espace en cellules dont le nombre croît linéairement et non pas exponentiellement avec la dimension. Citons aussi les approches qui permettent d'effectuer des recherches approximatives de plus proches voisins en introduisant de l'imprécision dans la recherche. L'idée est alors de réduire le temps de réponse en échange d'une imprécision contrôlée si possible [Ber04]. Cette méthode, comme toutes les autres méthodes de recherche de plus proches voisins, se compose de deux phases : une phase hors ligne de structuration des données, et une phase en ligne d'interrogation de la base. La phase hors ligne a pour objectif de regrouper les données en petits paquets les plus compacts et les plus séparés possibles. Ces propriétés de compacité et de séparabilité sont essentielles afin que les règles de filtrage utilisées par les algorithmes de recherche puissent être efficaces. L'algorithme utilisé pour la première phase est, selon Berrani, inspiré d'un algorithme de *clustering* appelé BIRCH, mais les modifications qu'il y apporte font plutôt penser au système DIGNET. Nous détaillerons ces approches par la suite.

4.3 Méthodes de *clustering*

Les méthodes de *clustering*, que l'on appelle aussi classification non supervisée, ont pour objectif de regrouper les données en différentes classes. Il est tout d'abord légitime de se demander ce qu'est une classe ? Bien qu'il n'y ait pas de définition universelle d'une classe, on décrit souvent la classe comme étant non seulement un ensemble homogène d'objets (qui ont donc des caractéristiques similaires), mais aussi un ensemble d'objets dont les caractéristiques diffèrent des autres classes.

Nous donnerons ici un aperçu des différents méthodes de classification non supervisées. Il existe bien entendu de nombreux articles sur ce sujet, si bien qu'il est impossible d'être exhaustif. Nous nous efforcerons donc de nous concentrer sur ce qui nous semble essentiel dans ce domaine. Il existe par ailleurs de nombreux articles de qualité qui offrent un aperçu dans ce large domaine. Citons par exemple [JMF99], [Ber02], [GCB04], [XW05] et [DHS00]. Nous nous en sommes largement inspiré quant à la méthodologie adoptée pour décrire le domaine de la classification non supervisée.

4.3.1 Notations et définitions

Considérons un ensemble d'objets $\mathcal{X} = \{x_1, ..., x_j, ..., x_N\}$, avec $x_j = (x_{j1}, ..., x_{jd})^T \in \mathbb{R}^d$ où x_{ij} est une caractéristique de l'objet x_j.

Suivant les algorithmes de classification utilisés, on aura des partitions de types différents :

- Les classifications de type *hard partition* cherchent à obtenir une partition $C = \{C_1, ..., C_K\}$ en $K \leq N$ classes de l'ensemble X, répondant aux critères :
  - $C_i \neq \varnothing$, avec $i = \{1, ..., K\}$
  - $\cup_{i=1}^{K} C_i = \mathcal{X}$
  - $C_i \cap C_j = \varnothing, \forall (i,j) \in \{1, ..., K\}$ et $i \neq j$
- Les classifications floues dérivent des classifications de type *hard partitions*, mais alors que pour les *hard partition*, chaque objet ne peut appartenir qu'à une seule classe, les algorithmes de classification flous autorisent les objets à avoir un degré d'appartenance $u_{i,j} \in [0, 1]$ pour chaque classe, où j représente le jème objet et i la ième classe. Deux contraintes doivent cependant être respectées :
  - $\forall j, \sum_{i=1}^{c} u_{i,j} = 1$
  - $\forall i, \sum_{j=1}^{N} u_{i,j} < N$
- Les classifications hiérarchiques construisent un ensemble $H = \{H_1, ..., H_Q\}$ avec $Q \leq N$ de partitions organisée de manière hiérarchique, répondant aux critères suivants :
  - $\forall i, j \neq i, m, l = \{1, ..., Q\ \}$
  - si $C_i \in H_m, C_j \in H_l$, et si $m > l$
  - alors $C_i \in C_j$ ou $C_i \cap C_j = \varnothing$

4.3.2 Distances

Le choix de la distance ou de la similarité est au cœur des algorithmes de classification et le choix de telle ou telle distance pourra influer sur la forme des classes. Rappelons tout d'abord ce qu'est une distance. Une distance $D(x_i, x_j)$ doit :

- être symétrique :

$$D(x_i, x_j) = D(x_j, x_i)$$

- être positive :

$$\forall x_i, x_j, D(x_i, x_y) > 0$$

- satisfaire l'inégalité triangulaire :

$$\forall x_i, x_j, x_k, D(x_i, x_j) \leq D(x_i, x_k) + D(x_k, x_j)$$

- être réflexive :

$$D(x_i, x_j) = 0 \text{ si et seulement si } x_i = x_j$$

Il existe plusieurs distances dans la littérature, citons en particulier :

- La distance de Minkowski :

$$D_{ij} = (\sum_{l=1}^{d} |x_{il} - x_{jl}|^{1/n})^n$$

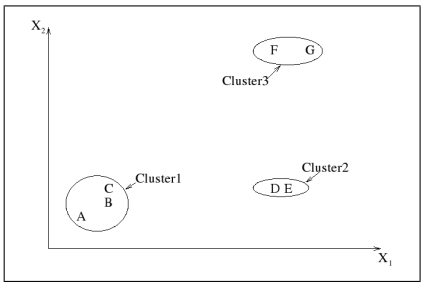


FIG. 4.1 – Données à classer.

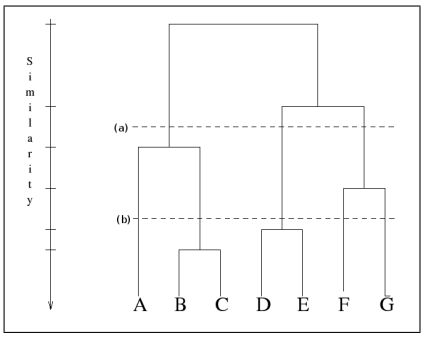


FIG. 4.2 – Le dendrogramme résultant.

– La distance euclidienne, qui est en réalité la distance de Minkowski avec $n = 2$:

$$D_{ij} = (\sum_{l=1}^{d} |x_{il} - x_{jl}|^{1/2})^2$$

– La distance City-block, qui est en réalité la distance de Minkowski avec $n = 1$:

$$D_{ij} = \sum_{l=1}^{d} |x_{il} - x_{jl}|$$

– La distance Sup, qui est en réalité la distance de Minkowski avec $n = \infty$:

$$D_{ij} = max_{1 \leq l \leq d} |x_{il} - x_{jl}|$$

– La distance de Mahanalobis :

$$D_{ij} = (x_i - x_j)^T S^{-1} (x_i - x_j)$$

où S est la matrice de covariance.

4.3.3 Les classifications hiérarchiques

4.3.3.1 Les méthodes classiques

La classification hiérarchique organise les données dans une structure hiérarchique appelée dendrogramme. Il existe deux grandes catégories de classification hiérarchique : celles qui sont agglomératives (de loin les plus courantes) et celles qui sont divisives.

Nous donnons, dans la Fig 4.2, le dendrogramme construit par une classification hiérarchique agglomérative à partir de données présentées en Fig 4.1. En coupant à différents endroits du dendrogramme, on obtient des partitions différentes. Par exemple, en coupant au niveau de a (voir Fig 4.2), on obtient comme partition $\{\{A, B, C\}; \{D, E\}; \{F, G\}\}$, alors qu'en coupant au niveau de b, on a $\{\{A\}; \{B, C\}; \{D, E\}; \{F\}; \{G\}\}$.

Nous donnons maintenant l'algorithme de construction du dendrogramme pour une classification hiérarchique agglomérative. Notons que dans le chapitre précédent nous nous sommes appuyés sur cette structure pour segmenter les symboles.

Initialisation des classes : chaque objet est placé dans une classe unitaire.

Soit $D(C_i, C_j)$ la distance de la classe C_i à la classe C_j, calculons la liste L des distances intra-classe.

Trions cette liste par ordre croissant.

(*) Enlever le premier élément de la liste qui correspond à la distance $d(C_n, C_m)$.

Fusionner les classes C_n et C_m.

Mettre à jour la liste des distances en tenant compte de la nouvelle classe $C_k = C_n \cup C_m$

Tant que la liste n'est pas vide ou que le critère ne convergence n'est pas atteint, retour en (*)

Algorithme 1: Pseudo algorithme pour la construction d'un dendrogramme.

Soit deux classes C_i, C_j contenant respectivement les objets $\{x_1^i, ..., x_{N_i}^i\}$ et $\{x_1^j, ..., x_{N_j}^j\}$ (où N_i et N_j sont respectivement le nombre d'objets dans la classe C_i et C_j). En fonction de la distance D choisie, on aura pour :

- $D(C_l, (C_i, C_j)) = min(D(C_l, C_i), D(C_l, C_j))$, la méthode single-link ([Sne57]).
- $D(C_l, (C_i, C_j)) = max(D(C_l, C_i), D(C_l, C_j))$, la méthode complete-link ([Sor48]).
- Que l'on peut généraliser en utilisant la formule donnée par Lance et Williams [LW67] :

$$D(C_l, (C_i, C_j)) = \alpha_i D(C_l, C_i) + \alpha_j D(C_l, C_j) + \beta D(C_i, C_j) + \gamma |D(C_l, C_i) - D(C_l, C_j)|$$

où $D(*, *)$ est une fonction distance et $\alpha_i, \alpha_j, \beta, \gamma$ sont des coefficients.

On peut trouver dans [Mur83], une liste de coefficients permettant de générer beaucoup d'autre distances, dont celle utilisant le minimum-variance (appelé distance de Ward) [War63], ou utilisant la moyenne des classes [JD88]. La distance choisie a un impact considérable sur les résultats. De récents travaux ([YW04]) cherchent toujours à trouver la distance la plus à même de faire resortir les différentes classes dans les données.

Les principales critiques sur les classifications hiérarchiques classiques sont :

- leur manque de robustesse, en particulier utilisées telles quelles, elles sont sensibles au bruit et aux outliers.
- leur complexité (le calcul de la matrice de proximité est en $O(n^2)$).
- leur tendance à former des classes ayant un aspect sphérique.

Nous allons donc maintenant voir les avancées effectuées dans ce domaine, qui tentent de pallier à l'un ou plusieurs de ces points.

4.3.3.2 BIRCH

C'est pour pallier la complexité trop élevée ainsi qu'au manque de robustesse par rapport au bruit des méthodes hiérarchiques classiques, que BIRCH [ZRL96] a été conçu. L'idée est de construire un arbre CF (Clustering Feature) regroupant les caractéristiques

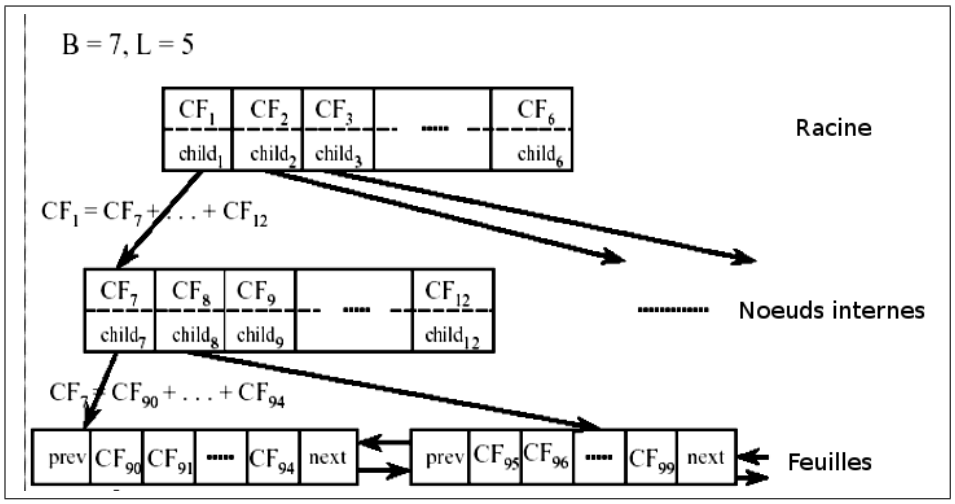


FIG. 4.3 – Un arbre CF (repris de [ZRL96]).

des différentes classes (voir Fig 4.3). Le vecteur CF_i qui est une représentation de la classe C_i est défini par :

$$CF_i = (N_i, LS, SS)$$

où $LS = \sum x_i$, $SS = \sum x_i^2$ et N_i est le nombre d'objets x_i dans le classe C_i.

Cette représentation permet de calculer facilement différentes caractéristiques pour chaque classe C_i :

– Le centroïd :

$$x_0^i = \frac{\sum_{k=1}^{d} x_k}{N_i}$$

– Le rayon :

$$R = \sqrt{\frac{\sum_{k=1}^{d}(x_k - x_0)^2}{N_i}}$$

– Le diamètre :

$$D = \sqrt{\frac{\sum_{k=1}^{d}\sum_{l=1}^{d}(x_k - x_l)^2}{N_i(N_i - 1)}}$$

On peut en déduire plusieurs mesures entre classes :

– D_0 : la distance euclidienne des centroïds :

$$D_0(C_i, C_j) = \sqrt{(x_0^i - x_0^j)^2}$$

– D_1 : la distance de Manahattan des centroïds :

$$D_1(C_i, C_j) = \sum_{k=1}^{d} |x_{0k}^i - x_{0k}^j|$$

– D_2 : la distance inter-classe moyenne :

$$D_2(C_i, C_j) = \sqrt{\frac{\sum_{k=1}^{N_i}\sum_{l=1}^{N_j}(x_k^i - x_l^j)^2}{N_i N_j}}$$

– D_3 : la distance intra-classe moyenne.
– D_4 : la distance variance croissante.

La construction de cet arbre nécessite 3 paramètres :
– B qui est le nombre maximum d'entrées pour un nœud interne
– L qui est le nombre maximum d'entrées pour une feuille
– T qui est diamètre maximum toléré pour chaque entrée des feuilles

Cet arbre se construit en y insérant un par un tous les objets à classifier. La procédure est décrite en Algo 2 :

Trouver la feuille de l'arbre appropriée : pour cela, descendre de manière récursive l'arbre CF, en choisissant à chaque fois le nœud enfant le plus proche en accord avec la mesure choisie ($D_0, D_1, D2, D_3$ ou D_4).

Modifier la feuille : Une fois arrivé à la feuille, trouver l'entrée L_i la plus proche, et regarder si L_i peut absorber l'objet sans violer le seuil maximum autorisé T.
Si (le seuil n'est pas dépassé) **Alors**
  réactualiser le vecteur CF de L_i
Sinon
  Si (nombre d'entrées inférieures à L) **Alors**
    ajouter une entrée correspondante à l'objet
  Fin Si
  diviser la feuille, en choisissant les 2 entrées les plus éloignées les unes des autres comme racine, et en redistribuant les entrées restantes selon le critère de proximité.
Fin Si
Modifier le chemin vers la feuille : Après avoir inséré l'objet dans la feuille, mettre à jour les vecteurs CF pour chaque chemin menant à la feuille qui vient d'être modifiée.
Si (on n'a pas fait de divisions) **Alors**
  ajouter l'objet au vecteur CF
Sinon
  Si (nombre d'entrées non feuille < B (*)) **Alors**
    mettre à jour le vecteur CF
  Sinon
    diviser le nœud parent retour en (*)
  Fin Si
Fin Si

Algorithme 2: Pseudo algorithme d'insertion d'un objet dans l'arbre CF (BIRCH).

Une fois l'arbre créé, on peut appliquer une classification hiérarchique classique sur l'ensemble des sommaires de l'arbre. La complexité résultante devient alors $O(n)$.

4.3.3.3 CURE

CURE [GRS98] a été spécialement développé pour pouvoir identifier correctement des classes de forme quelconque. L'idée est d'associer non pas un représentant par classe (comme on le fait habituellement avec le centroïd ou le médoïd), mais un ensemble de représentants qui doivent pouvoir représenter la classe de manière suffisamment correcte. Soit C_i une classe regroupant un ensemble d'objet, on lui attache les caractéristiques suivantes :

- $C_i.mean$, qui représente la moyenne des objets présents dans la classe C_i.
- $C_i.rep$, qui est un ensemble d'objets qui représentent la classe C_i.

L'algorithme va donc être identique aux classifications classiques, mis à part la définition de $D(C_l, (C_i, C_j))$. Posons $C_w = C_i \cup C_j$. On a alors $C_w.mean = \frac{N_i C_i.mean + N_j C_j.mean}{N_i + N_j}$.

Le calcul de $C_w.rep$ est donné en Algo 3. On a en particulier deux paramètres à fixer :

- c qui est le nombre de représentants dans $C_w.rep$
- $\alpha \in [0, 1]$ qui est là pour rapprocher ou éloigner les représentants de la moyenne de la classe. En particuler, si on prend $\alpha = 0$, cela revient à faire une classification hiérarchique classique.

tmpSet= $\emptyset$
Pour k **de** 1 **à** c **faire**
  maxDist = 0
  Pour m **de** 1 **à** N_w **faire**
    Si (k=1) **Alors**
      $minDist = d(C_w.mean, x_m^w)$
    Sinon
      $minDist = min_q d(x_m^w, x^q)$ avec $x^q \in$ tmpSet
    Fin Si
    Si (minDist $\geq$ maxDist) **Alors**
      $maxDist = minDist$
      $maxX = x_m^w$
    Fin Si
    $tmpSet = tmpSet \cup x_m^w$
    $C_w.rep = \emptyset$
    Pour chaque $x^q \in tmpSet$ **faire**
      $C_w.rep = C_w.rep \cup \{x^q + \alpha(C_w.rep - x^q)\}$
    Fin Pour
  Fin Pour
Fin Pour

Algorithme 3: Calcul de $C_w.rep$.

4.3.4 Les classifications minimisant une erreur

Les classifications qui utilisent l'erreur carrée ne permettent d'avoir en sortie qu'une seule partition. Celle-ci est trouvée en optimisant une fonction définie localement ou globalement. La plus utilisée est :

$$E = \sum_{i=1}^{K} \sum_{j=1}^{N_i} d(x_j^i, m_i)^2$$

avec K le nombre de classes désirées, N_i le nombre de données contenues dans C_i, m_i le centre de la classe C_i, et d une distance (euclidienne par exemple). Malheureusement, il est impossible de parcourir toutes les combinaisons possibles. En effet, pour N données à organiser en K groupes, il y a $\frac{K^N}{K!}$ énumérations possibles. Ce qui donne pour 20 objets et 3 groupes à faire, pratiquement 600 millions de possibilités ! Plusieurs méthodes ont donc vu le jour pour essayer de pallier ce problème.

4.3.4.1 K-means

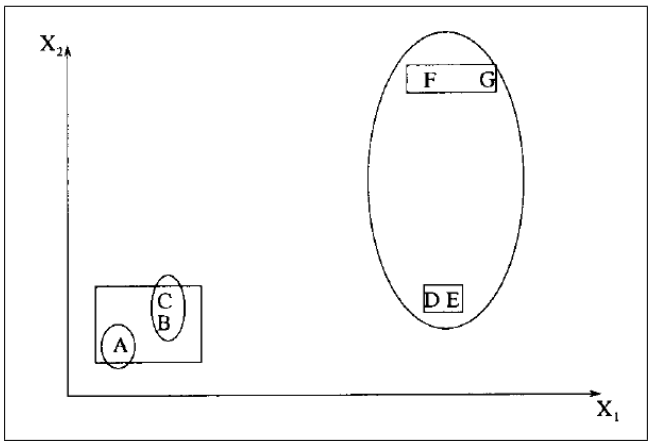


FIG. 4.4 – Résultats obtenus avec les Kmeans pour différentes initialisations.

La méthode des K-means [McQ67] est décrit en Algo 4. Cet algorithme est universellement utilisé notamment à cause de sa faible complexité en $O(N)$. Pourtant il a plusieurs inconvénients. Le premier est qu'il n'y a pas de méthode pour trouver de manière efficace le nombre de classes K. Mais ce problème est inhérent à pratiquement toutes les méthodes de classification non supervisée. Le second problème est la dépendance des résultats au choix de la partition initiale. La figure 4.4 en donne un exemple. Si l'on souhaite trouver 3 classes, et que l'on choisit comme tête de classe les objets $\{A, B, C\}$, on obtient au final $\{\{A\}, \{B, C\}, \{D, E, F, G\}\}$. Par contre si l'on choisit au début les objets $\{A, D, F\}$, on obtient $\{\{A, B, C\}, \{D, E\}, \{F, G\}\}$. Différentes méthodes ont été proposé pour remédier à ce problème, l'objectif globale étant de trouver une partition initiale plus appropriée et relancer l'algorithme plusieurs fois en ne gardant que la meilleure solution [KR90].

k : **nombre de classes désirées**
Choisir de manière aléatoire k objets qui représenteront les k classes
Répéter
  Calculer la moyenne $m_i, i \in [1, k]$ de chaque classe C_i
  Pour chaque $x_n, n \in [1, N]$ **faire**
    Trouver le $i \in [1, k]$ qui minimise $\| x_n - m_i \|$
    Assigner x_n à la classe C_i
  Fin Pour
jusqu'à ce que (les classes C_i ne changent plus)

Algorithme 4: Pseudo algorithme des K-means.

4.3.4.2 PAM

Pour éviter une trop forte sensibilité au bruit, on peut choisir comme représentant d'un classe non pas un centroïd mais un medoïd. C'est ce que PAM [KR90] (Partitioning Around Medoid) s'emploie à faire (cf Algo 5). Il cherche parmi tous les objets à classer, les k meilleurs medoïds. Il sépare les objets entre ceux qui sont sélectionnés O_i (et qui sont pour l'instant les candidats à être les medoïds), et les objets qui ne le sont pas O_h. A chaque itération, il calcule pour chaque couple (O_i,O_h) une mesure TC_{ih} qui rend compte de la pertinence de remplacer O_i par O_h. Si cette mesure est positive, on a une amélioration, si elle est négative, il n'est pas pertinent de les permuter.

k : **nombre de classes désirées**
O_i : **les objets sélectionnés**
O_h : **les objets non-sélectionnés**

Choisir de manière aléatoire k objets qui représenteront les k classes
Répéter
  Calculer TC_{ih} pour toutes les paires d'objets O_i et 0_h
  Sélectionner la paire (O_i, O_h) qui minimise TC_{ih}
  Si $(minTC_{ih} > 0)$ **Alors**
    Remplacer O_i par O_h
  Fin Si
jusqu'à ce que $(minTC_{ih} > 0)$
Assigner chaque objet O_h à la classe C_i qui minimise $d(O_h, O_i)$

Algorithme 5: Pseudo algorithme PAM.

Si PAM donne de bons résultats sur de petits ensembles, il est néanmoins inapplicable

sur de plus grands ensembles à cause de sa complexité élevée en $O(k(n-k)^2)$.

4.3.4.3 CLARA

C'est pour pallier à la complexité trop grande de PAM que CLARA [KR90] (Clustering LARge Applications) a été développée. L'idée est d'appliquer l'algorithme PAM non pas sur la totalité des données mais sur une partie seulement choisie aléatoirement. De plus, pour éviter d'être trop dépendant de la partie aléatoire, ce procédé est répété plusieurs fois. Ainsi les auteurs proposent de prendre 5 échantillons aléatoire de taille $40 + 2k$ où k est le nombre de classes désiré. CLARA est décrit en algo 6.

Pour j **de** 1 **à** 5 **faire**
  Prendre un échantillon aléatoire des données de taille $40 + 2k$
  Exécuter l'algorithme PAM sur cet échantillon
  Assigner chaque objet O_h à la classe C_i qui minimise $d(O_h, O_i)$
  Calculer la dissimilarité moyenne de la classification obtenue

Fin Pour
Ne retenir que la partition qui a obtenu la plus faible dissimilarité moyenne

Algorithme 6: Pseudo algorithme CLARA

Cet algorithme a alors une complexité en $O(k(40+k)^2 + k(n-k))$. Ce qui le rend plus adapté pour de larges bases de données.

4.3.4.4 CLARANS

Les algorithmes PAM et CLARA peuvent être vu comme une recherche dans un graphe. Ce graphe noté $G_{n,k}$, a des nœuds qui sont représentés par un ensemble d'objets $\{O_{m_1}, ..., O_{m_k}\}$ qui sont les k médoïds choisis. Deux nœuds sont connectés si les ensembles d'objets auxquels ils sont rattachés ne diffèrent que d'un élément. De plus à chaque nœud est associé un coût qui représente la dissimilarité totale. On voit donc que PAM revient à rechercher le nœud au coût minimum dans $G_{n,k}$. Alors que CLARA va rechercher un minimum dans un sous graphe de $G_{n,k}$. CLARANS [NH02] (Clustering Large Applications based upon RANdomize Search) se propose donc de rechercher le minimum dans le graphe complet $G_{n,k}$, mais en maintenant l'idée de CLARA de ne prendre en compte qu'un échantillonnage des données. Cependant cet échantillon sera pris sur l'ensemble des voisins du nœud à traiter. CLARANS est décrit en Algo 7.

numlocal : **le nombre maximal de minimaux locaux trouvés autorisés**
tailleVoisinage : **la taille du voisinage à considérer**

```
i=1
coutMini=∞
Tant que (i<numlocal) faire
    Choisir de manière aléatoire un nœud noté current de G_{n,k}
    j=1
    Tant que (j<tailleVoisinage) faire
        Prend un échantillon du voisinage de current que l'on note S
        Si (cout(S)<coutMini) Alors
            current=S
            j=1
        Sinon
            j=j+1
        Fin Si
    Fait
    [Si on est sorti de la boucle, alors cela veut dire que l'on a trouvé un minimum local]
    [Si ce minimum est le plus faible coût déjà vu, alors on le garde en mémoire]
    Si (coût(current)<coutMini) Alors
        coutMini=coût(current)
        bestNode=current
    Fin Si
    i=i+1
Fait
Retourner bestNode
```

Algorithme 7: Pseudo algorithme CLARANS.

4.3.4.5 ISODATA

ISODATA [BH67] (Iterative Self-Organizing Data Analysis Techniques) est aussi un algorithme très populaire. Contrairement aux K-means, le nombre de classes n'est pas fixé définitivement, puisqu'il va automatiquement s'ajuster en autorisant des fusions avec les classes similaires, et des divisions avec les classes qui ont une trop grande variabilité. De plus, il permet de s'affranchir des outliers (cf Algo 8).

Définissons les variables suivantes :
- K est le nombre de classes désirées,
- I le nombre maximum d'itérations,
- P le nombre maximum de fusions autorisées,
- θ_N le nombre minimum d'éléments qu'une classe doit contenir pour ne pas être rejetée,

- θ_S le seuil au delà duquel on considère que la déviation dans une classe est trop grande (et donc division de cette classe),
- θ_C le seuil en dessous duquel on considère que la similarité doit conduire à la fusion des deux classes concernées.

Choisir de manière aléatoire K centre de classes m_i.
Pour n **de** 1 **à** I **faire**
  Assigner les objets x_i à la classe C_j qui minimise $d(x_i, m_j)$
  Eliminer les classes qui ont un nombre d'éléments insuffisants (ie $N_{C_j} < \theta_N$)
  Recalculer les centres de classes m_i.
  Pour chaque classe, calculer la distance moyenne du centre de classe aux objets. $D_i = \frac{1}{N_i} \sum_{x_j \in C_i} d(x_j, m_j)$
  Calculer la distance globale moyenne des objets à leur centre de classe. $D = \frac{1}{N} \sum_{i=1}^{N} N_{C_i} D_i$
  Si $(k \leq K/2)$ **Alors**
    [On a trop peu de classes, donc on va diviser des classes]
    On calcule la déviation standard $\sigma^{(j)}$ pour chaque classe C_j
    Pour chaque classe, on trouve la composante maximale $\sigma_{max}^{(j)}$ de $\sigma^{(j)}$
    Pour j **de** 1 **à** k **faire**
      Si $(\sigma_{max}^{(j)} < \sigma^(j)$ ou $D_j > D$ ou $N_j > 2\theta_N)$ **Alors**
        On divise le classe C_j.
      Fin Si
    Fin Pour
  Fin Si
  Si $(k > 2K)$ **Alors**
    [On a trop de classes, donc on va fusionner des classes]
    Calculer les distances D_{ij} de similarité entre la classe C_i et la classe C_j
    Trouver au plus les P plus faibles valeurs de D_{ij}
    Fusionner les classes correspondantes
  Fin Si
Fin Pour

Algorithme 8: Pseudo algorithme ISODATA

ISODATA est donc beaucoup plus flexible au niveau du nombre de classes à trouver, mais c'est au prix de nombreux paramètres supplémentaires à régler savamment.

4.3.4.6 Les classifications probabilistes

D'un point de vue probabiliste, les données à classifier peuvent être vues comme des données générées par des distributions probabilistes. Ainsi, si l'on connaît la nature des distributions (gaussienne ou autres), les données peuvent nous aider à estimer les para-

mètres de ces distributions et donc, par la suite, être classifiée. Les hypothèses de départ sont donc que les données sont générées par une distribution de paramètres θ_i avec une probabilité τ_i ($i \in [1, K]$, avec K le nombre de classes que l'on souhaite obtenir). La probabilité que l'on génère x connaissant θ est donnée par :

$$p(x|\theta) = \sum_{i=1}^{K} \tau_i p(x|C_i, \theta_i)$$

avec $\theta = (\theta_1, ..., \theta_K)$, $\sum_{i=1}^{K} \tau_i = 1$ et $p(x|C_i, \theta_i)$ étant la probabilité que x soit généré par la distribution i connaissant θ_i. On utilise alors l'estimation du maximum de vraisemblance où la vraisemblance est :

$$l(\theta) = \sum_{j=1}^{N} logp(x_j|\theta)$$

La meilleure estimation de θ est alors trouvée en cherchant la solution à l'équation suivante :

$$\frac{\delta\theta}{\delta\theta_i} = 0$$

Comme cette équation peut rarement être résolue de manière analytique, il faut alors utiliser des méthodes itératives sous-optimales, parmi lesquelles l'algorithme EM (expectation-maximization) [MK97] est le plus populaire. Cet algorithme est en deux grandes étapes (voir Algo 9) :

initialiser θ^0
t=0
Tant que (pas de convergence) **faire**
    étape E : calcul de la vraisemblance $l(\theta^t)$ sur l'ensemble des données
    étape M : recherche d'un nouveau paramètre θ^{t+1} qui maximise l
Fait

Algorithme 9: Pseudo algorithme EM

On répète ces étapes jusqu'à ce que l'on atteigne une convergence. Parmi les inconvénients de cette méthode on peut citer :

- la sensibilité aux paramètres initiaux,
- la possibilité d'avoir une matrice de covariance singulière qui mettra en défaut la méthode,
- la possibilité de converger vers un minimum local,
- la lenteur de la convergence.

Il a été montré qu'une utilisation de la classification EM avec des gaussiennes revenait à utiliser les K-means [CG92].

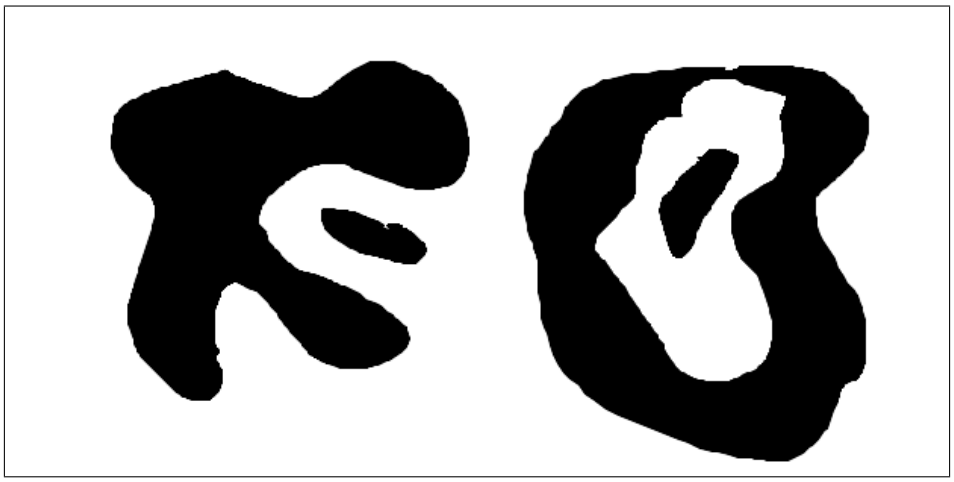


FIG. 4.5 – Classes ayant des formes irrégulières.

4.3.5 Les classifications basées sur la densité

Une autre manière de voir la classification est de la considérer à travers la notion de densité. De ce point de vue, une classe n'est rien d'autre qu'un ensemble d'objets suffisamment proches les uns des autres. On peut alors détecter des classes qui ont des formes irrégulières (Fig 4.5), ce qui est très délicat avec des classifications de types K-means.

4.3.5.1 DBSCAN

Les concepts qui sont en jeu dans ce type de méthode sont la densité et la connectivité. L'algorithme DBSCAN [EKSX96] (Density Based Spatial Clustering of Applications with Noise) repose sur deux paramètres ϵ et $MinPts$. Des définitions sont nécéssaires pour comprendre la signification de ces termes et de l'algorithme.

- Le ϵ-voisinage d'un objet x est défini par $N_\epsilon(x) = \{y \in X | d(x,y) \leq \epsilon \}$.
- On appelle objet cœur, un objet possédant plus de $MinPts$ objets dans son voisinage.
- Un objet y est densité-atteignable à un objet cœur x, si il existe une série d'objets cœur entre x et y, de telle manière que chacun appartienne à l'ϵ-voisinage de l'autre.
- Deux points x et y sont densité-connectés s'ils sont densité-atteignables à un objet cœur commun.

L'idée est de repérer les ensembles qui sont suffisamment denses, puis de les regrouper quand ils sont connectés. Un algorithme succint est donné en Algo 10.

Tant que (Il existe des objets non classés) **faire**
  Prendre un objet x non classé au hasard.
  Si (x est un objet cœur) **Alors**
    Trouver tous les objets y densité-atteignable à x.
    Les classer dans la même classe que x.
  Sinon
    Classer x dans une classe Bruit.
  Fin Si
Fait
Vérifier que les objets de la classe Bruit ne sont pas densité-connecté aux objets cœur précédemment trouvés.

Algorithme 10: Pseudo algorithme DBSCAN

Avec une telle méthode, nous n'avons pas à fixer un nombre de classes, et de plus nous pouvons trouver des classes ayant des formes complexes. Cependant, il est difficile d'ajuster les paramètres ϵ et $MinPts$. De plus, la méthode ne gère pas le cas où l'on aurait des classes de densités différentes. On peut aussi noter que la notion de densité rend cette méthode inapplicable aux données de grandes dimensions et ceci à cause du phénomène d'espace vide.

4.3.5.2 OPTICS

C'est pour pallier le fait que DBSCAN ne peut trouver des classes de densités différents, et ce à cause du fait que ϵ et $MinPts$ sont fixés une fois pour toutes, que OPTICS [ABKJ99] (Ordering Points To Identify the Clustering Structure) a été développé. Cette méthode s'appuie largement sur DBSCAN. Cependant au lieu d'utiliser une seule valeur pour ϵ, elle va travailler sur une plage de valeurs $[\epsilon_1, \epsilon_2]$. De plus, lors de la construction de la classe, elle garde pour chaque objet la distance minimale de cet objet à un objet cœur. En traçant un graphique dont l'abcisse représente les différents objets considérés lors de l'algorithme et dont l'ordonnée représente la distance minimale de cet objet à un cœur objet, on peut alors facilement retrouver la structure des données (voir Fig 4.6).

4.3.5.3 DBSCLAD

DBSCLAD [XEKS98] (Distribution Based Clustering of Large Spatial Databases) s'attaque aussi au problème des classes ayant des densités différentes mais avec une approche probabiliste. Cette méthode émet l'hypothèse que les données de chaque classe sont uniformément distribuées (ce qui n'est pas toujours le cas), et va se baser sur un test χ^2 pour vérifier cette hypothèse. Un algorithme simplifié est proposé en Algo 11.

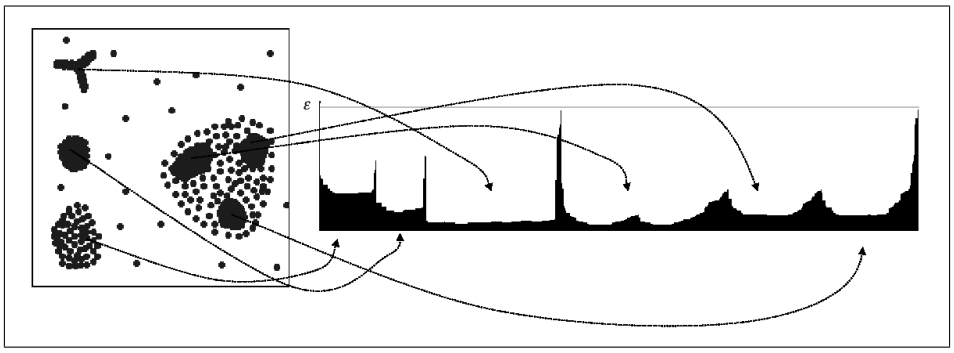


FIG. 4.6 – OPTICS (repris de [ABKJ99]).

Tant que (Il existe des objets non classés) **faire**
  Prendre un objet x non classé au hasard.
  Créer une classe C et y mettre l'objet x
  Ajouter à la liste des candidats pour la classe C (les 29 objets les plus proches de x).
  Pour Chaque candidat y à la classe C **faire**
    Ajouter à la liste des candidats pour la classe C, les objets se trouvant dans une rayon bien défini par rapport à y
  Fin Pour
  Pour Chaque candidat y à la classe C **faire**
    Ajouter y à la classe C.
    Si (on a pas la bonne distribution) **Alors**
      On retire y de la classe C.
    Fin Si
  Fin Pour
Fait

Algorithme 11: Pseudo algorithme DBSCLAD

Le rayon est calculé automatiquement et est fonction de l'aire de la classe et de son nombre d'objets. L'avantage de cette méthode est qu'elle ne nécessite aucun paramètre, mais elle est 2 à 3 fois moins rapide que DBSCAN, ce qui n'est pas si dramatique surtout si on considère le fait que DBSCAN doit être lancé plusieurs fois dans le but de trouver les meilleurs valeurs des paramètres.

4.3.5.4 DENCLUE

DENCLUE [HK98] (DENsity-based CLUstEring) se base sur des fonctions de densité $f^D(x)$. En chaque point x est calculé une valeur réelle en fonction du point considéré et

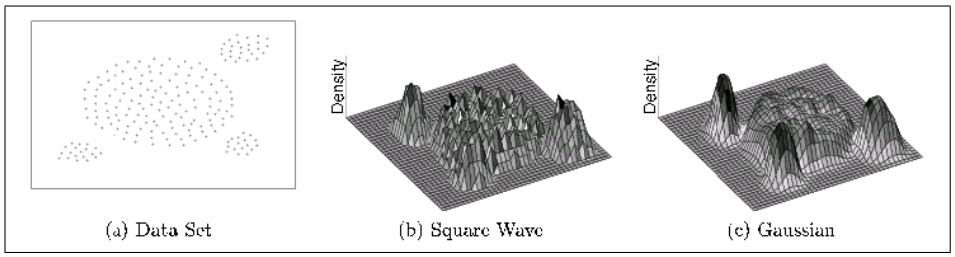


FIG. 4.7 – (a) Les données et les résultats de la fonction densité : (b) avec la porte, (c) avec la gaussienne (repris de [HK98]).

de ses voisins. Cette fonction est la superposition de fonctions influences.

$$f^D(x) = \sum_{y \in D} f(x, y)$$

Plusieurs types de fonctions peuvent être utilisées. De type gaussienne :

$$f(x, y) = e^{-\frac{d(x,y)^2}{2\sigma^2}}$$

De type fonction porte :

$$f(x, y) = \begin{cases} 1 & \text{si } d(x, y) < \sigma \\ 0 & \text{sinon} \end{cases}$$

La figure 4.7 montre ce que ces fonctions peuvent donner. Pour des raisons de complexité, on ne considère pas tous les points voisins de x , mais seulement les plus proches. Ainsi on a $D = y/d(x, y) < k\sigma$. Ensuite les maxima locaux sont détectés avec un méthode de descente de gradient. On ne considère que les maxima supérieurs à un seuil ϵ donné. DENCLUE est une méthode générique, puisque qu'en utilisant la fonction porte cela conduit aux k-means, alors qu'avec la fonction gaussienne cela revient à l'algorithme DBSCLAD.

4.3.6 Les classifications se basant sur la théorie des graphes

Nous avons vu comment les classifications pouvaient être formalisées en utilisant les graphes. Les données sont alors représentées par un graphe pondéré. Chaque nœud représente un objet, et les arcs nous indiquent la dissimilarité entre les 2 objets qu'ils connectent. Intéressons-nous maintenant aux classifications qui se basent sur de tels graphes.

4.3.6.1 MST

Une idée fort simple est d'utiliser l'arbre de longueur minimale [Zah71]. Comme illustré sur la figure 4.8, il suffit de supprimer successivement les arcs. Le critère d'arrêt peut être soit le nombre de classes (qui est représenté par le nombre de sous-graphes que l'on a générés en enlevant les arcs), soit la longueur minimale en dessous de laquelle il n'est plus pertinent de supprimer des arcs.

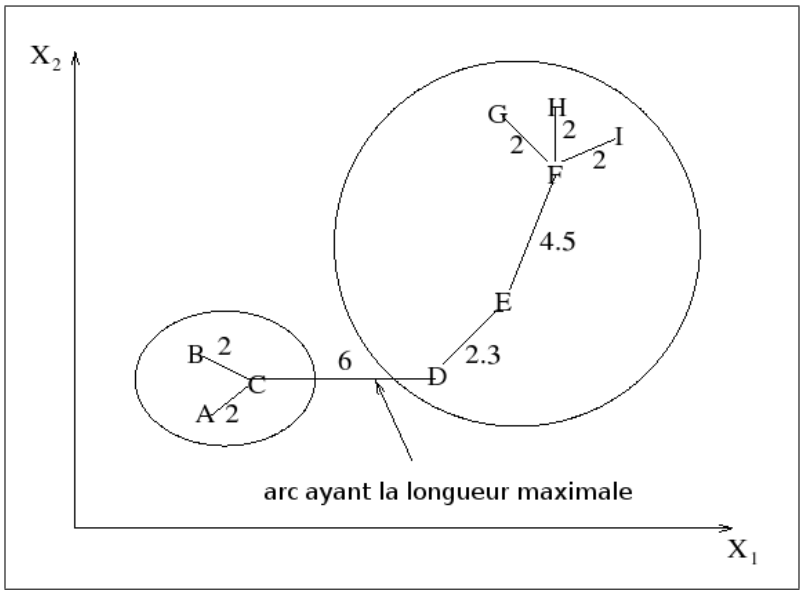


FIG. 4.8 – Arbre de longueur minimale.

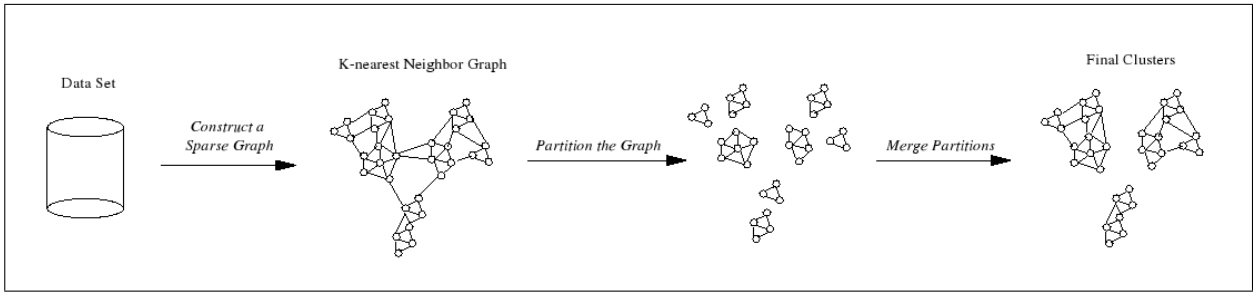


FIG. 4.9 – CHAMELEON (repris de [KHK99]).

4.3.6.2 CHAMELEON

Le principe de CHAMELEON [KHK99] est de ne fusionner les classes uniquement si elles sont proches en terme de distance et si elles ont la même interconnectivité. La figure 4.9 décrit le fonctionnement global.

A partir des données, on va d'abord construire le graphe des k plus proches voisins. Ce graphe va être ensuite partitionné de telle manière que l'on ne garde que des groupes d'objets fortement liés entre eux. Il en résulte un très grand nombre de classes. Ces classes vont ensuite être fusionnées en tenant compte de deux paramètres : leur proximité relative et leur interconnectivité.

4.3.6.3 CLICK

Une autre méthode pour partitionner un graphe est d'utiliser une méthode qui tente d'isoler des sous-graphes très connectés (c'est-à-dire des sous-graphes dont le nombre

d'arcs nécessaires pour isoler ce sous-graphe est supérieur à 0.5 fois le nombre de nœuds). Pour cela, on applique de manière récursive une procédure qui isole des sous-graphes en enlevant le moins d'arcs possibles [HS00].

CLICK [SS00] est un peu basé sur ce même principe. Cependant dans cette méthode, les arcs ont un poids e_{ij} qui interviennent dans le partitionnement du graphe.

$$e_{ij} = log \frac{Proba(\text{i et j appartiennent au même cluster})|S_{ij}}{Proba(\text{i et j n'appartiennent pas au même cluster}|S_{ij})}$$

où S_{ij} représente la dissimilarité entre les objets i et j.

4.3.7 Les classifications floues

4.3.7.1 FCM

FCM [Bez81] (Fuzzy C-Means) est la plus courante des classifications floues. Elle tente de trouver une partition en c classes qui minimise une fonction coût :

$$J(U, M) = \sum_{i=1}^{c} \sum_{j=1}^{N} u_{i,j}^{m} D_{i,j}$$

où
- U= $[u_{i,j}]$ est une matrice avec $u_{i,j} \in [0, 1]$ représentant l'appartenance de l'objet x_j à la classe C_i.
- M=[m_1,...,m_c] représente les prototypes des classes C_i
- m est le paramètre flou qui vaut usuellement 2.
- D_{ij} la distance entre l'objet x_j et le prototype m_i.

L'algorithme est décrit ci-dessous.

Choisir c et m
Initialiser M aléatoirement
t=0
Répéter
  Pour chaque u_{ij} de U **faire**
    $u_{ij}^{t+1} = 1/\sum_{l=1}^{c} (\frac{D_{lj}}{D_{ij}})^{1/(1-m)}$
  Fin Pour
  Pour i **de** 1 **à** c **faire**
    $m_i^{t+1} = (\sum_{j=1}^{N} (u_{ij}^{t+1})^m x_j)/(\sum_{j=1}^{N} (u_{ij}^{t+1})^m)$
  Fin Pour
jusqu'à ce que ($\| M^{t+1} - M^t \| \leq \epsilon$)

Algorithme 12: FCM

Les principaux inconvénients sont :
– la sensibilité au bruit et aux outliers.
– la difficulté de trouver une bonne partition initiale

4.3.7.2 PCM

PCM [KK93] modifie la fonction coût afin d'être robuste aux outliers.

$$J(U,M) = \sum_{i=1}^{c}\sum_{j=1}^{N} u_{i,j}^{m} D_{i,j} + \sum_{i=1}^{c} n_i \sum_{j=1}^{N} (1-u_{i,j})^m$$

avec $n_i > 0$. De cette manière on donne plus de poids aux objets qui ont une forte appartenance aux classes, et donc l'effet des outliers est amoindri.

4.3.7.3 RCA

Frigui [FK99] proposent avec RCA (Robust Competitive Agglomeration) une méthode issue des statistiques robustes, de plus ils se basent dans le cas où le nombre de classes c est inconnu. Reprenons la fonction coût initiale (avec m=2) :

$$J(U,M) = \sum_{i=1}^{c}\sum_{j=1}^{N} u_{i,j}^{2} D_{i,j}$$

Cette fonction J est a son minimum lorsque $c = N$, ce qui n'est pas pratique si l'on veut trouver automatiquement c (on aurait systématiquement c=N). Mais on peut rajouter un terme de régularisation pour éviter ce phénomène, on a alors :

$$J_A(U,M) = \sum_{i=1}^{c}\sum_{j=1}^{N} u_{i,j}^{2} D_{i,j} - \alpha \sum_{i=1}^{c} (\sum_{j=1}^{N} u_{i,j})^2$$

où α est un paramètre à fixer. Comme le premier terme contient toujours une fonction aux moindres carrés, cette formulation n'est pas robuste. En effet, un outlier aura un impact beaucoup trop important à cause de la distance carrée que l'on prend. Il faut alors prendre :

$$J_R(U,M) = \sum_{i=1}^{c}\sum_{j=1}^{N} u_{i,j}^{2} \rho_i D_{i,j} - \alpha \sum_{i=1}^{c} (\sum_{j=1}^{N} w_{ij} u_{i,j})^2$$

avec ρ_i une fonction robuste associée à la classe C_i, et $w_{ij} = \frac{\delta \rho_i D_{ij}}{\delta D_(ij)}$ représente une fonction poids. Ces fonctions sont choisies par l'utilisateur en fonction du type de classes qu'il désire retrouver.

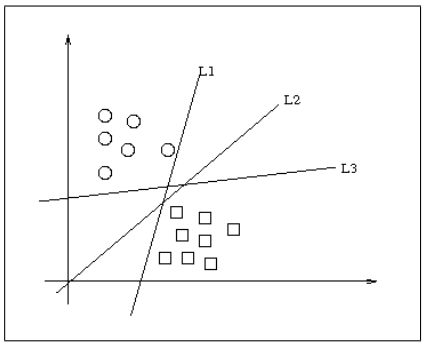


FIG. 4.10 – Différents hyperplans séparant deux populations.

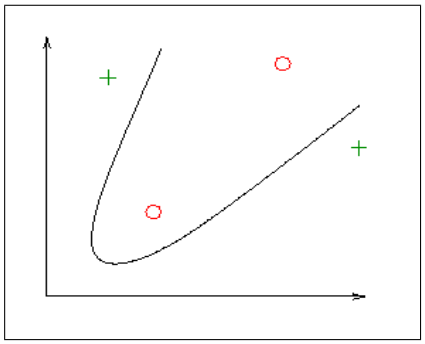


FIG. 4.11 – Séparation non linéaire de deux populations.

4.3.8 Autres types de classification

Il existe encore de nombreuses méthodes de classification qui n'entrent pas dans les catégories que nous venons de voir, nous en donnons ici un bref aperçu.

4.3.8.1 Support Vector Machine (SVM)

Les SVM ont été introduits par Vapnik [Vap98]. Considérant deux populations dans un espace de dimension donné, le problème est de trouver un hyperplan qui permette de séparer ces populations. Comme le montre la figure 4.10, il peut exister différents hyperplans répondant à ce critère, mais les SVM s'attache à trouver l'hyperplan qui va maximiser la distance du point le plus proche à l'hyperplan (appelé marge). Quand il n'est pas possible de séparer les populations avec une droite (dans un espace de dimension de 2), il est possible de passer à un espace de représentation intermédiaire de plus grande dimension. Puis à l'aide d'un noyau qui n'est rien d'autre qu'une fonction qui matérialise une distance adapté au problème, on peut alors séparer les populations 4.11.

4.3.8.2 Les classifications à base de d'algorithme de recherche

Dans la plupart des classifications, l'enjeu est de minimiser une fonction coût. Comme rechercher la meilleure partition de manière déterministe est prohibitif en terme de temps

de calcul, on est amené à rechercher une partition sous-optimale avec si possible la convergence la plus rapide [RKD89]. Le SA (Simulated Annealing, recuit simulé en français) [KGV83] a donc été conçu dans cet objectif. Cependant, afin d'éviter les extremums locaux, cet algorithme va accepter avec une certaine probabilité que la solution suivante soit moins performante. Cette probabilité se traduit en terme de température. Plus la température est élevée, plus cette probabilité sera grande (cf Algo 13).

Fixer la température initiale T_0 et la température finale T_f (T_0>T_f)
Choisir une partition P_0 de manière aléatoire.
Calculer son coût E_{P_0}.
$T = T_0$
Tant que (T<T_f) **faire**
    Pour i **de** 1 **à** nbIteration **faire**
        choisir un voisin P_1 de P_0
        calculer E_{P_1}
        Si ($E_{P_1} > E_{P_0}$) **Alors**
            $E_{P_0} = E_{P_1}$ avec une certaine probabilité T
        Sinon
            $E_{P_0} = E_{P_1}$
        Fin Si
    Fin Pour
    $T = cT$
Fait

Algorithme 13: Pseudo algorithme du recuit simulé.

nbIteration représente le nombre de recherche que l'on fait à une température T donnée. c est une valeur comprise entre 0 et 1 qui va faire décroître progressivement la température. Statistiquement cette méthode garantit de trouver la solution globale, mais la convergence peut être longue, car la température doit alors descendre de manière très progressive.

4.3.8.3 Algorithme génétique

Les algorithmes génétiques tentent d'appliquer ce que l'on connaît de la théorie de l'évolution. On a, à la base un certaine population qui va évoluer de génération en génération. Cette évolution se fait à l'aide de trois opérateurs :

- la sélection : les plus adaptés ont plus de chances de passer à la génération suivante.
- la recombinaison : la génération suivante est issue d'un couplage entre deux populations issues de la génération précédente.
- la mutation : la génération suivante est la conséquence d'une altération d'une population de la génération précédente.

Un des premiers algorithmes utilisant cette technique est l'algorithme génétique [Hol75]. Les populations sont représentées par N chaînes binaires $c_i, i \in [1, N]$. Par exemple, la chaîne binaire 11010011 signifie que les éléments 1, 2, 4, 7 et 8 appartiennent à la classe numéro 1. A cette chaîne binaire est associée une valeur calculée à l'aide d'une fonction qui rend compte de la pertinence de cette population. Un algorithme succint est donné en Algo 14.

Choisir la taille N de la population que l'on veut considérer.
Choisir une population de manière aléatoire.
Tant que (pas de convergence) **faire**
    Calculer pour chaque population $f(c_i)$.
    Utiliser les opérateur d'évolutions avec une certaine probabilité pour générer la prochaine génération.
Fait

Algorithme 14: Pseudo algorithme génétique.

Cet algorithme a été utilisé pour la première fois par [RB79]. Pour une classification de N objets en K classes, les partitions sont représentées par une chaîne K-ary. Par exemple, si l'on a 6 objets A, B, C, D, E, F à classer en 2 classes, les partitions seront représentées par une chaîne binaire de taille 6. Ainsi la chaîne 100111 représente la partition A,D,E,F,B,C.

L'avantage de ce type de méthode est qu'elle se présente comme une recherche globale de la solution optimale. En effet, l'opérateur de mutation permet de générer des solutions qui sont complètement différentes des précédentes. Ces mèthodes possèdent néanmoins plusieurs inconvénients, comme la sensibilité et la difficulté de fixer les différents paramètres comme la taille N de la population à considérer, la probabilité que les opérateurs de mutation et de recombinaison interviennent. De plus, elles souffrent également d'une complexité assez élevée.

4.3.8.4 Algorithme à base de noyau

Utiliser la décomposition en valeur singulière [MR95] permet une autre approche de classification. Le PDDP (Principal Direction Divise Partitioning) [Bol98] a permis de faire une avancée intéressante dans la classification de documents. L'idée est de diviser successivement l'espace des données en deux, ce qui donne un arbre binaire.

Imaginons que nous ayons N documents x_i à classer et que nous disposions d'un lexique de M mots y_j. On va construire une matrice $m_{i,j}$ qui représente le nombre d'occurrences du mot y_i dans le document x_i. Cette matrice est ensuite décomposée en valeur singulière et, en ne retenant que la plus grande, on peut alors diviser l'espace des documents en deux. Puis, on peut réitérer. Les résultats obtenus semblent prometteurs.

4.3.8.5 Combinaison de méthodes

On voit de plus en plus se développer de méthodes hybrides qui tentent de ne prendre que les avantages de certaines méthodes tout en minimisant l'impact des inconvénients inhérents à celle-ci. L'approche la plus répandue consiste à utiliser une méthode minimisant une erreur, dont le principal avantage est la faible complexité. Puis d'utiliser à la suite une méthode hiérarchique, qui sera plus à même de nous renseigner sur le nombre de classes présent. Le principal inconvénient des méthodes hiérarchiques étant leur grande complexité, celui-ci est évité car on n'injectera qu'une partie des données en entrée ([FL03]).

4.3.9 Les classifications pour de grandes bases de données

Classer de grandes bases de données soulève des problèmes en terme de temps de calcul et de mémoire disponible. C'est un champ de recherche très actif et différentes méthodes ont vu le jour pour résoudre cet épineux problème. En effet, la plupart des méthodes que nous avons vues sont incapables de passer à l'échelle. Les classifications hiérarchiques classiques par exemple ont une complexité en $O(n^2)$ et la mémoire requise est tout simplement phénoménale dans le cas de grandes bases de données.

4.3.9.1 Classification incrémentale

Thomopoulos ([TB91]) propose le système DIGNET inspiré des approches neuronales pour traiter le problème des grandes bases de données. Avec DIGNET chaque donnée est examinée tour à tour. Si cette donnée est proche d'un centre de classe existant, ce centre de classe est renforcé (cf théorie des neurones). Par contre s'il est assez loin des centres de classes existants, on crée alors un nouveau centre de classe (on ajoute un neurone). Ainsi, en un seul passage la classification est faite. La place mémoire utilisée est très réduite puisqu'on ne considère qu'une seule donnée à la fois, et de plus cette méthode est robuste aux outliers. Cependant, la classification obtenue dépend fortement de l'ordre dans lequel les données sont apparues.

4.3.9.2 Réduction de données

Une autre technique pour gérer les grandes bases de données est de parcourir les données et d'essayer d'extraire des sommaires. BIRCH que nous avons déjà vu en est un exemple. L'arbre CF construit capture les données récurrentes. Il est à noter que cette phase d'extraction des sommaires a été étendue aux attributs de catégories ([CFCW01]). Alors que BIRCH repose sur un espace vectoriel pour construire ses sommaires, Ganti ([GRG+99]) propose un arbre BUBBLE reposant sur le même principe mais qui fonctionne dans un espace métrique. Ainsi chaque feuille de l'arbre BUBBLE est composée :

- du nombre de points ,
- du medoïd,
- de la distance moyenne des points de la feuille au medoïd.

4.3.9.3 Echantillonage

Pour gérer les très grandes bases données on peut faire appel aux techniques d'échantillonage qui vont extraire une partie seulement de cette base de données (en les prenant au hasard par exemple). Cet échantillonnage va donc lui aussi permettre de passer à l'échelle. Les méthodes de classifications anciennes y ont fait appel avec plus ou moins de rigueur statistique (CLARANS, classification fractale, k-means).

Une avancée notable dans ce domaine réside sans aucun doute dans les travaux concernant la borne de Chernoff ([NGC99]) qui stipule qu'indépendamment de la distribution d'une variable aléatoire à valeur réelle Y (Avec $0 < Y < R$), la moyenne de n observations indépendantes est comprise dans :

$$|Y - \frac{1}{n}\sum_{i=1}^{n} Y_i| \leq \epsilon$$

avec une probabilité de $1 - \delta$, et avec

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}$$

Ces bornes ont notamment été utilisées dans l'algorithme CURE.

4.3.10 Les classifications pour les données de grandes dimensions

L'autre grand problème est de traiter des données de grandes dimensions. Nous avons déjà donné quelques-unes de ces raisons précédemment. De plus dans les espaces de grandes dimensions, la distance des plus proches voisins devient instable, et il est alors impossible de distinguer les vrais proches voisins, de ceux qui ne le devraient pas. Beyer [BGRS99] a montré qu'à partir de 15 dimensions, la notion de proximité commençait déjà à être mise en défaut.

4.3.10.1 Réduction des dimensions

Une première méthode est de réduire la dimension des données, ce qui permet par la suite de traiter le problème avec des méthodes plus classiques. L'analyse en composante principale ([MKJ80]) en est un des plus célèbres exemples, mais qui conduit souvent à des classes difficilement interprétables. Les techniques de décomposition par valeur (SVD) sont aussi couramment utilisées dans le même but.

4.3.10.2 Classification en sous-espace

L'algorithme CLIQUE (CLustering In QUEst [AGGR98]), repose sur la définition de rectangle unitaire. On ne retient que les rectangles qui ont une densité supérieure à σ.

Ces rectangles se construisent avec une approche bottom-up. Les rectangles en dimension 1 sont trouvés en divisant l'espace en grille de largeur δ (σ et δ étant donné par l'utilisateur). On passe de manière récursive des rectangles de dimension $q-1$ aux rectangles de dimension q en fusionnant les rectangles ayant au moins les $q-2$ premières dimensions identiques. Ensuite les sous-espaces sont triés par densité, et les moins denses sont éliminés. Cette méthode permet d'avoir une vision beaucoup plus riche que la classification en partitions puisqu'elle permet de voir des classes en différents sous-espaces.

L'algorithme MAFIA (Merging of Adaptive Finite IntervAls [NGC99]) est une extension de CLIQUE qui permet d'avoir un échantillonnage différent selon les dimensions, et qui fusionne les rectangles de dimension $q-1$ qui ont au moins $q-2$ dimensions communes (et pas seulement les $q-2$ premières).

4.3.11 Combien de classes ?

Dans de nombreuses méthodes, le nombre de classes K désiré doit être donné par l'utilisateur. Malheureusement, ce nombre est rarement connu, et il va sans dire que la qualité de la classification en est grandement dépendant. Plusieurs méthodes ont été développées pour tenter de trouver ce nombre.

4.3.11.1 Visualisation des données

Une première méthode est de tout simplement pouvoir visualiser les données afin que l'utilisateur puisse estimer à vue d'oeil le nombre de classes présentes dans les données. Si cela est relativement simple en dimension 2, cela commence à se compliquer en dimension 3, sans parler des dimensions supérieures...

4.3.11.2 Construction d'indices

La construction d'indices a pour objectif de mettre en valeur ce que l'on considère comme étant une bonne classification, c'est à dire une forte homogénéïté dans les classes, et une forte disparité entre les classes. Une évaluation des 30 indices [MC85] a montré qu'un des meilleurs indices était celui présenté par Caliñski et Harabasz [CH74] :

$$CH(K) = \frac{Tr(S_B)}{K-1} / \frac{Tr(S_W)}{N-K}$$

avec N, le nombre total d'objets, $Tr(S_B)$ et $Tr(S_w)$ la trace de la matrice inter et intra classe. Le K maximisant $CH(K)$ étant le nombre de classes optimal.

Parmis les travaux plus récents on peut citer le coefficient de Silhouette [KR90] décrit ci-après. Soit un point x dans une classe C, on peut calculer deux mesures $a(x)$ et $b(x)$:

$$a(x) = \frac{1}{N_C} \sum_{y \in C, y \neq x} d(x,y)$$

$$b(x) = min_{G \neq C} \frac{1}{G} \sum_{y \in G} d(x, y)$$

Le coefficient de Silhouette est alors défini par $s(x) = (b(x) - a(x))/max(a(x), b(x))$. Quand ce coefficient est proche de 1 alors, le point est dans la bonne classe. En calculant une moyenne des $s(x)$, on peut alors avoir une idée globale de la qualité de la classification.

D'une manière générale, il n'y a pas d'indices nettement supérieurs à tous les autres, et bien souvent la qualité de ces indices dépendent des données.

4.3.11.3 Optimisation du nombre de classes

Les classifications de type probabiliste se prêtent tout à fait à un autre type de méthode pour trouver le nombre de classes qui utilisent leur spécificité. On a par exemple :

- Le critère d'information de Akaike (AIC) [Boz83] :

  $$AIC(K) = \frac{-2(N - 1 - N_k - K/2)l(\theta)}{N} + 3N_p$$

  avec N le nombre total de classes, N_k le nombre d'objets dans chaque classe C_k, N_p le nombre de paramètres qui on été estimés et $l(\theta)$ le logarithme du maximum de vraisemblance. On choisit le K qui minimise AIK(K).
- Le critère Bayesien d'information (BIC) [Sch78] :

  $$BIC(K) = l(\theta) - (N_p/2)log(N)$$

  On choisit le K qui maximise BIC(K).
- La longueur de description mimimum (MDL) [Ris78] :

  $$MDL(K) = l(\theta) + N_p/2log(N)$$

  On choisit le K qui minime MDL(K).
- On peut encore aussi citer : la longueur du message minimum (MML) [WF87], le critère théorique d'information non codée (ICOMP) [Boz94],...

La pertinence de ces critères dépend fortement du cadre d'utilisation, de sorte qu'il est impossible de sortir un critère du lot.

4.3.12 Conclusion

Dans la figure 4.12, nous représentons les grandes familles de méthodes ou les grandes méthodes, et nous indiquons pour chacune d'elle, la complexité (si elle est connue), la forme des *clusters* générés, ainsi que les avantages et inconvénients que l'on peut citer. Il en ressort que le choix d'une méthode de *clustering* dépend surtout de l'application que l'on veut en faire.

Dans notre cas, qui est l'indexation d'une grande base de données, nous avons besoin d'une méthode qui a une faible complexité, qui soit robuste au bruit et qui donne des

Méthode	complexité	forme des *clusters*	avantages	inconvénients
hiérarchique	$O(N^2)$	sphérique	structure hiéarachiques	peu robuste aux outliers
BIRCH	$O(N)$	sphérique	robuste au bruit	
CURE	faible	quelconque	robuste au bruit	
KMean	$O(N)$	sphérique	simple d'utilisation	dépend de la partition initiale
EM	élevé	sphérique	issue de théories probabilistes	possibilité maximum local
DBSCAN	$O(NlogN)$	quelconque	basé sur la notion de densité	paramètres à régler
DENCLUE	$O(NlogN)$	quelconque	robuste au bruit	
FCM	$O(N^2)$	sphérique	issue de théorie floue	peu robuste aux outliers
RCA	élévé	sphérique	issue des statistiques robustes	

FIG. 4.12 – Récapitulatif sur les différentes méthodes de *clustering*

clusters de formes sphériques (ceci afin de limiter au maximum le chevauchement entre les classes qui viendrait détériorer les performances des règles de filtrage). C'est pourquoi le choix s'est porté sur un algorithme de type BIRCH. Nous allons donc adaptater la méthode de Berrani [Ber04], ce qui fera donc l'objet du prochain chapitre. Puis nous verrons comment nous pouvons augmenter les performances de l'algorithme en combinant différents descripteurs.

Chapitre 5

Méthode proposée

5.1 Introduction

Nous avons vu dans le chapitre précédent différentes méthodes pour indexer une base de données multidimensionnelle. Nous nous proposons maintenant d'en voir une qui nous a particulièrement intéressé. Nous verrons donc en détail les paramètres que cette méthode utilise, puis nous proposerons une adaptation originale.

5.2 Algorithme utilisé

Afin de regrouper de manière efficace les données, nous nous avons adapté un algorithme mis au point par Berrani [Ber04]. Notre choix s'est porté sur cet algorithme car il obtient des résultats tout à fait intéressant quand il traite de très large base de données, aussi bien en terme de temps de calcul que de qualité d'indexation ce qui le rend tout à fait adapté à notre problème. On peut assimiler cet algorithme à une fusion entre la première phase de l'algorithme BIRCH (cf 4.3.3.2) et l'algorithme DIGNET (cf 4.3.9.1). Nous en donnons ici le fonctionnement.

Cette première phase de BIRCH est une opération de micro-*clustering* dont le but est de regrouper les vecteurs de la base en petits paquets et d'identifier les vecteurs isolés. Le but de cette opération est donc de résumer la base de données à l'aide des centres des micro-*clusters*. Initialement, chaque vecteur forme un micro-*cluster* auquel on associe un CF-vecteur défini par $CF = (N, LS, r)$, où N est le nombre de vecteurs présents dans ce micro-*cluster*, LS est un vecteur qui est la somme des vecteurs présents, et r le rayon de l'hypersphère englobante de l'ensemble des vecteurs.

Ainsi lorsque deux micro-*clusters* fusionnent on a :

$$CF_1 + CF_2 = (N_1 + N_2, LS_1 + LS_2, max(d(c, c_1), d(c, c_2)))$$

avec c le centre de gravité du micro-*cluster* résultant, et c_1 et c_2 les centres de gravité

respectivement du premier et du second micro-*cluster*. Le centre de gravité se calcule aisément avec $c = \frac{1}{N}.LS$.

V : **liste de vecteurs à regrouper**
nbMaxClust : **nombre maximum de *clusters* à créer**
β : **taux de bruit**
t_0 : **valeur initiale du paramètre** t
$listeTmp$: **liste des CF-vecteurs des micros-*clusters***
$listeTmp$: **liste des CF-vecteurs des micros-*clusters* bruités**

$listeTmp = V$
$t = t_0$

Tant que (taille de listeTmp < nbMaxClust) **faire**
  Vider($listeTmp$)
  Pour i **de** 1 **à** taille de listeTmp **faire**
    Si (taille de $listeTmp = 0$) **Alors**
      Insérer le micro-*cluster* $listeTmp[i]$ dans $listeTmp$)
    Sinon
      Retrouver p, l'indice du micro-*cluster* de $listeTmp$ le plus proche de $listeTmp[i]$
      Calculer r_{fus}, le rayon de la fusion éventuelle entre $listeTmp[i]$ et $listeTmp[p]$
      Si ($r_{fus} > t$) **Alors**
        Insérer le micro-*cluster* $listeTmp[i]$ dans $listeTmp$
      Sinon
        Fusionner les micros-*clusters* $listeTmp[i]$ et $listeTmp[p]$
      Fin Si
    Fin Si
  Fin Pour
  AbsorberBruit($listeTmp$,$listeTmpBruit$,t)
  IsolerBruit($listeTmp$,$listeTmpBruit$,β)
  Incrémenter t
  Vider($listeTmp$)
  $listeTmp = listeTmp$
Fait

Algorithme 15: Regroupement par paquets

Soit un seuil t donné, l'idée ici est de parcourir tous les micro-*clusters* de la base un par un. A chaque micro-*cluster* rencontré, on cherche le micro-*clusters* le plus proche. Deux cas peuvent alors se présenter, soit la distance est inférieure au seuil t auquel cas les deux micro-*clusters* fusionnent, soit la distance à la classe est supérieure au seuil et dans

ce cas le micro-*cluster* est laissé intact.

Une fois que l'on a passé en revue tous les micro-*clusters*, on regarde les moins peuplés que l'on classe comme bruit. Ceci, afin de gérer au mieux les phénomènes d'outlier. Puis, le seuil est progressivement augmenté jusqu'à ce qu'on ait atteint le nombre de micro-*clusters* voulu (cf Algo 15). Dans cet algorithme, la fonction $absorberBruit()$ permet de regarder si les micro-*clusters* qui ont été considérés comme bruités à l'itération précédente, ne peuvent pas maintenant être réintégrés dans un des micro-*clusters* courants. La fonction $IsolerBruit()$ permet, quant à elle, de repérer dans la liste des micro-*clusters* courants, les micro-*clusters* qui ne sont pas assez peuplés et qui vont donc être maintenant considérés comme du bruit.

5.2.1 Les différents paramètres

Différents paramètres sont donc nécessaire au bon fonctionnement de cet algorithme. Nous étudions donc maintenant étudier chacun de ces paramètres afin de pouvoir les fixer au mieux.

5.2.1.1 Concernant le bruit

Nous avons vu que pour qu'un micro-*cluster* soit déclaré comme étant du bruit, il faut qu'il soit moins peuplé comparativement aux autres. Cette procédure est la même qui est utilisée dans l'algorithme de classification BIRCH. C'est pour cela que nous allons reprendre les paramètres que les auteurs préconisent, à savoir : le micro-*cluster* est classé comme bruit, s'il contient moins de $\beta\%$ par rapport à la moyenne faite sur l'ensemble des classes avec $\beta = 15\%$.

5.2.1.2 Le critère d'arrêt

Berrani ([Ber04]) fixe comme critère d'arrêt un nombre de micro-*cluster* minimum en dessous duquel il faut stopper l'algorithme. Ainsi, au départ on a de nombreux micro-*clusters*, puis au fur et à mesure que le seuil est augmenté, le nombre de micro-*cluster* diminue. Nous fixons alors comme seuil $\delta\sqrt{N}$ où N est le nombre total de vecteurs, et δ un paramètre à fixer que nous étudierons par la suite.

5.2.1.3 Incrémentation du seuil

Un autre paramètre à fixer est la manière dont on va incrémenter le seuil t. On comprend bien, que si on augmente trop rapidement ce paramètre, la qualité du regroupement en sera affecté, il faut donc augmenter progressivement ce paramètre sans que cela puisse pour autant être trop lent, car nous manipulons de larges bases de données.

Soit N_a le nombre de classes suffisant pour arrêter les itérations. Soit N_i, le nombre de classes correspondant au seuil t_i. Pour déterminer t_{i+1}, nous allons faire une approximation

linéaire en partant du principe que nous voulons à l'itération suivante $N_{i+1} = \alpha N_i$. Ainsi on a :

$$t_{i+1} = \alpha \frac{N_i - b}{a}$$

avec :

$$a = \frac{N_i - N_{i-1}}{t_i - t_{i-1}}$$

et :

$$b = \frac{t_i N_{i-1} - t_{i-1} N_i}{t_i - t_{i-1}}$$

5.2.1.4 Initialisation du seuil

Pour cet algorithme il reste à initialiser t_0. Si certains préconisent d'utiliser $t_0 = 0$, cela ne nous semble pas judicieux. En effet, utiliser $t_0 = 0$ revient à calculer au départ toutes les distances deux à deux entre les données, ce qui est très pénalisant dans notre cas, car nous utilisons de large bases de données. Afin de pallier ce problème, nous avons mis en place un système qui permet d'initialiser t_0, en se basant sur une estimation des distances qui sont en jeu.

Pour ce faire, nous choisissons de manière aléatoire un fraction des distances entre chaque donnée (dans notre cas 1% des distances). Les distances calculées sont ensuite triées par ordre croissant, puis nous choisissons comme valeur initiale la η ième valeur sur le nombre total de distances calculées. Ainsi $\eta = 0.5$ correspond au choix de la valeur médiane sur le pourcentage de distance calculée.

5.2.2 Critères de filtrage

Une fois que les paquets sont formés, vient la recherche proprement dite des k plus proches voisins. Cette recherche s'effectue à l'aide de deux règles de filtrage. Soit une requête q, et $C_i, i \in [0, N_i]$ (avec N_i le nombre de paquets) les paquets résultant du regroupement des données. r_i représente le rayon associé au paquet C_i, et m_i son centre de classe. On utilise aussi d_{max_i} et d_{min_i} qui sont définis par :

$$d_{max_i} = d(q, m_i) + r_i$$

$$d_{min_i} = d(q, m_i) - r_i$$

où $d(x, y)$ est une distance entre x et y.

Le première règle de filtrage permet de mettre de côté les paquets non pertinents en se basant sur leur distance par rapport à la requête. Ainsi, cette règle suit les étapes suivantes :

1. calculer la distance d_{max_i} de chaque paquet C_i contenant au moins k vecteurs. Les paquets ne répondant pas à cette condition ne sont pour l'instant pas considérés avec d_{max_m} la plus petite de ses distances
2. calculer la distance d_{min_i} sur tous les paquets C_i. Si $d_{min_i} > d_{max_m}$, on peut alors écarter le paquet C_i.

Pour illustrer cette règle regardons les figures 5.1 à 5.4. Imaginons que nous avons des données qui ont été regroupé en 6 paquets, et que nous recherchons les 4 plus proches voisins d'une requête q (fig 5.1). On calcule en premier la distance d_{max} de la requête au différents paquets (fig 5.2). On choisit ensuite la plus petite que l'on appelle d_{max_m}. Ensuite on calcule la distance d_{min} de la requête au paquet C_1. Comme cette distance est plus grande que d_{max_m}, on écarte le paquet C_1 dans son ensemble (fig 5.3). On parcourt ensuite les autres paquets, au final il ne reste plus que 2 paquets (fig 5.4).

La deuxième règle de filtrage suit quant à elle les étapes suivantes :

1. trier les paquets par d_{min} croissant.
2. examiner le contenu de tous les paquets un par un.
3. arrêter la recherche si $d_{min_i} \geq d(q, kkpv_c)$, avec $kppv_c$ le k^{ime} plus proches voisins courants.

Si nous reprenons notre exemple précédent (Fig 5.5 et 5.6). Après avoir examiné le paquet C_4 qui avait une distance d_min plus faible que le paquet C_6, on fait ressortir le 4ème plus proche voisin courant (noté $d(q, kkpv_c)$ sur la Fig 5.5). Puis on regarde le paquet C_6. Ce paquet ayant une distance d_{min} supérieur à $d(q, kkpv_c)$, il n'est alors pas nécessaire de regarder les données contenues dans le paquets C_6, et la recherche peut s'arrêter.

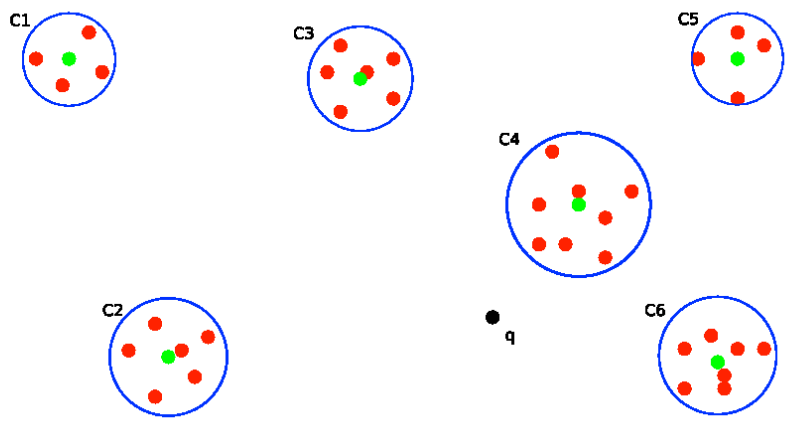


FIG. 5.1 – Règle filtrage 1 : Données initiales.

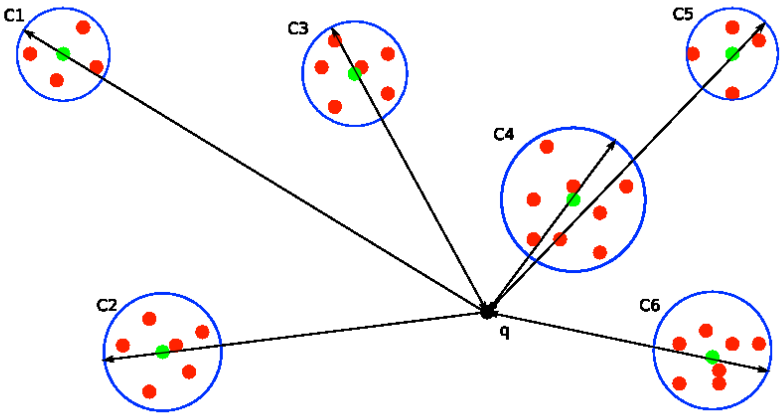


FIG. 5.2 – Règle filtrage 1 : Calcul des d_{max}.

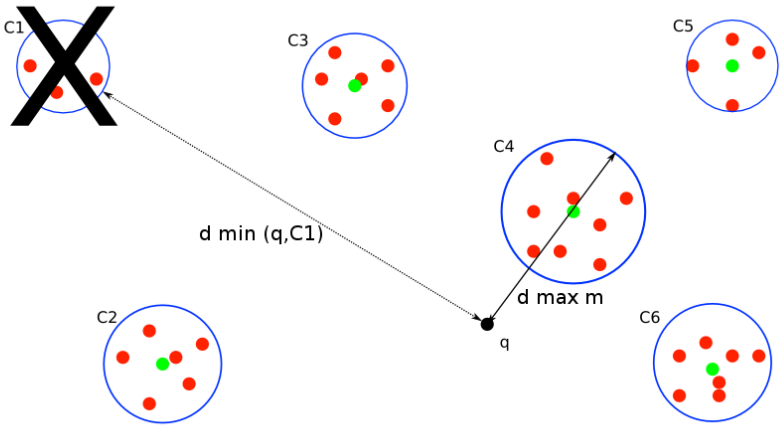


FIG. 5.3 – Règle filtrage 1 : Filtrage du paquet C_1.

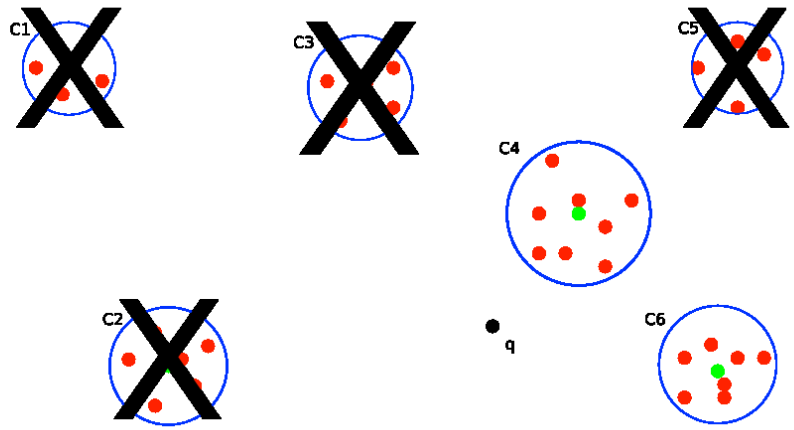


FIG. 5.4 – Règle filtrage 1 : Résultat de la première règle de filtrage.

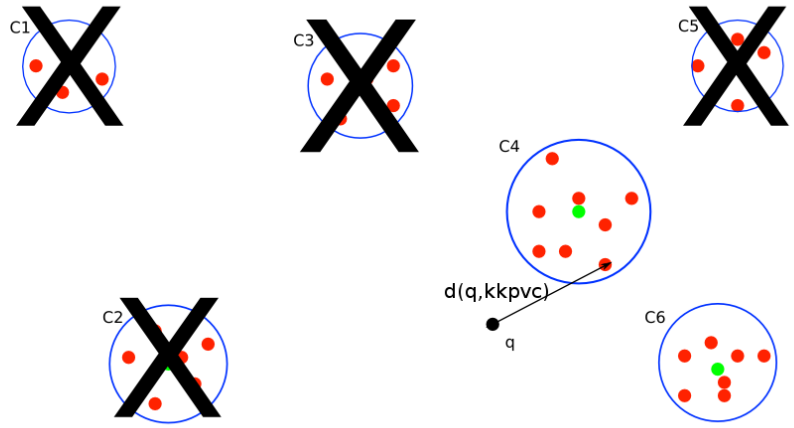


FIG. 5.5 – Règle filtrage 2 : Calcul du 4ème plus proche voisins courant.

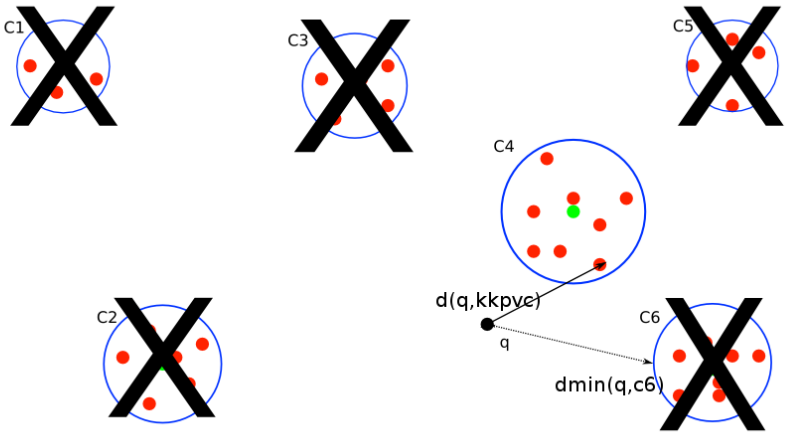


FIG. 5.6 – Règle filtrage 2 : Filtrage du paquet C_6.

5.2.3 Evaluation des paramètres

L'évaluation des paramètres ne peut se faire ici qu'en considérant les temps de recherche des k plus proches voisins. Ainsi pour évaluer les paramètres évoqués en 5.2.1, nous avons repris les données issues de l'expérimentation, à savoir les 200000 symboles qui étaient répartis dans 100 plans techniques. Les paramètres à évaluer sont l'initialisation du seuil t_0 à travers du paramètre η, l'incrémentation du seuil t à travers du paramètre α, le critère d'arrêt à travers du paramètre δ ainsi que le taux concernant le bruit β. Pour commencer nous avons choisi comme valeur : $\alpha = 0.5$, $\eta = 0.5$, $\beta = 0.15$ et $\delta = 2$.

Initialisation du seuil. Les figures 5.7 et 5.8 montrent l'évolution du temps de réponse en fonction du temps de calcul de l'algorithme, en faisant varier η de 0.1 à 0.7 par pas de 0.1. On s'aperçoit que d'un part avec η=0.7 on s'approche du temps de réponse que l'on avait avec la recherche séquentielle et exhaustive qui était pour rappel de 18s. Par contre ce temps est divisé par deux avec η=0.4. Au delà, le temps de réponse ne bouge quasiment plus, mais par contre le temps de calcul, lui, augmente de manière conséquente. Il nous paraît alors raisonnable de choisir η=0.4 comme paramètre par défaut.

Incrémentation. En gardant $\eta = 0.4$, nous choisissons maintenant ensuite d'étudier l'incidence de α. Les figures 5.9 et 5.10 montrent l'évolution du temps de réponse et du temps de calcul de l'algorithme, en faisant varier α de 0.2 à 0.7 par pas de 0.1. On s'apperçoit alors que les temps, aussi bien de calcul que de recherche, sont stables. Ceci est surprenant car on aurait pu s'attendre à ce que les temps de calcul pour une faible incrémentation soient plus importants. Ceci est du au fait que l'approximation linéaire n'est pas parfaite. En effet, pour tous les α que nous avons fixés, la taille successive des paquets n'a que faiblement variés. Dans notre cas avec $\eta = 0.4$, l'algorithme trouve lors de sa première itération environ 3000 paquets. Alors qu'avec $\eta = 0.2$, on s'attend à avoir

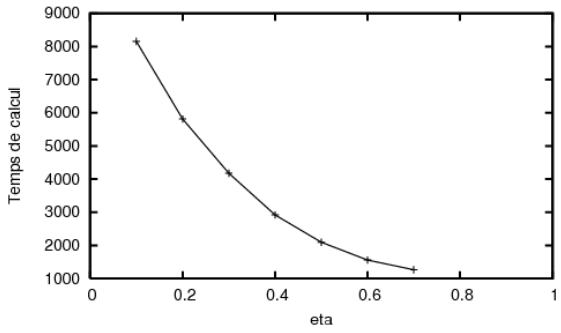


FIG. 5.7 – Temps de calcul (en s) pour différentes valeurs de η.

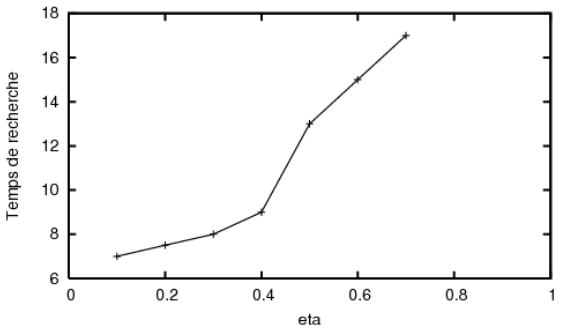


FIG. 5.8 – Temps de réponse (en s) pour différentes valeurs de η.

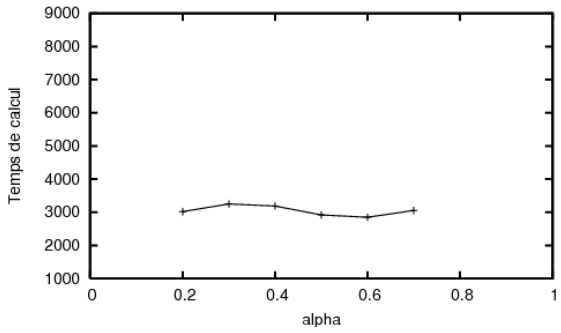


FIG. 5.9 – Temps de calcul (en s) pour différentes valeurs de α.

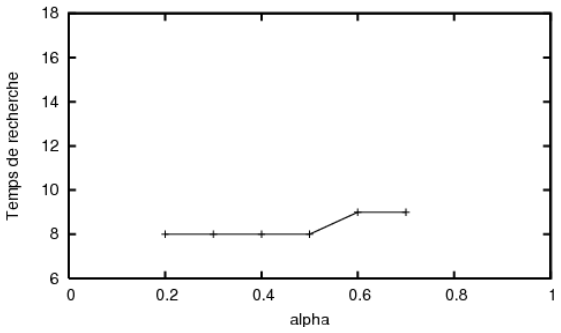


FIG. 5.10 – Temps de réponse (en s) pour différentes valeurs de α.

un rayon qui nous donne $3000(1 - 0.2) = 2400$ paquets, ce rayon nous donne en réalité 1500 paquets. On a un comportement identique pour toutes les valeurs de η que nous avons fixées. On en déduit donc, dans notre cas, que η n'a qu'une influence limitée. Nous choisissons donc pour la suite de le fixer à 0.5.

Critère d'arrêt. Dans cette partie nous voulons vérifier le fait que les performances de l'algorithme sont stables sur une plage assez large de valeur. Les figures 5.11 et 5.12 montrent l'évolution du temps de réponse et du temps de calcul de l'algorithme, en faisant varier δ de 1 à 4 par pas de 1. On s'aperçoit alors que le critère d'arrêt est peu sensible dans cette plage de valeur. Nous choisissons alors de prendre $\delta = 2$.

Conclusion Il résulte de cette étude que les paramètres qui nous semblent optimaux sont : $\alpha = 0.4$, $\eta = 0.5$, $\beta = 15\%$ et $\delta = 2$. Avec ces paramètres nous avons un temps de calcul hors-ligne d'environ 50min (3000s), et un temps de réponse de 8s.

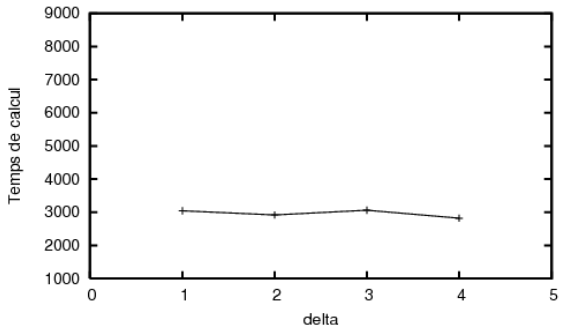


FIG. 5.11 – Temps de calcul (en s) pour différentes valeurs de δ.

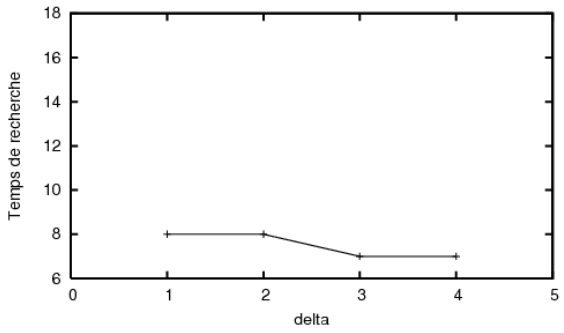


FIG. 5.12 – Temps de réponse (en s) pour différentes valeurs de δ.

5.3 Améliorations proposées

Alors que les récentes approches tentent de réduire le temps de recherche en ne cherchant que les plus proches voisins approximatifs [Ber04], nous avons décidé d'essayer une nouvelle approche propre à la recherche de symboles. En effet, à partir d'un symbole il est facile de calculer différents types de descripteurs. Nous avons vu pour l'instant que l'ART que nous employons était assez performant. Imaginons maintenant que nous puissions réduire le temps de recherche en utilisant d'autres descripteurs qui nous permettraient d'élaguer rapidement les paquets non pertinents.

5.3.1 Présentation de la méthode

Prenons un descripteur simple comme l'ellipticité. Ce descripteur est vraiment peu performant en terme de recherche, en effet, il est inimaginable dans la majorité des cas, de faire une recherche en utilisant uniquement ce descripteur. Mais l'avantage d'un tel descripteur est qu'il varie peu aux possibles déformations que peut subir le symbole. L'idée est alors de calculer pour tous les symboles candidats un tel descripteur. Les valeurs qui en résultent sont alors triées par ordre croissant, ce qui nous permet ensuite de diviser cette distribution de valeurs en n paquets de tailles sensiblement équivalentes. Ces paquets font ensuite l'objet d'un regroupement via le système présenté précédemment, en utilisant les paramètres que nous avons établis comme optimaux et en utilisant le descripteur ART.

Prenons le cas concret de l'ellipticité. En étudiant la distribution des valeurs de l'ellipticité sur l'ensemble des valeurs, nous sommes capables de diviser cette distribution en disons 20 paquets de tailles équivalentes. Ainsi lorsque l'on a une requête à présenter, il suffit de ne considérer que les paquets les plus pertinents. A condition, bien sûr, d'être capable de disposer de bornes qui nous permettent de dire quels sont effectivement les paquets pertinents. Ces bornes sont propres aux descripteurs que nous choisissons. Ainsi l'ellipticité étant une valeur comprise entre 0 et 1, il semble raisonnable de penser que si la requête a une ellipticité e_q, chercher les paquets ayant une ellipticité e telle que $|e_q - e| \leq 0.2$ nous permet d'élaguer les paquets qui ne sont pas pertinents. La figure 5.13

illustre le principe de la méthode.

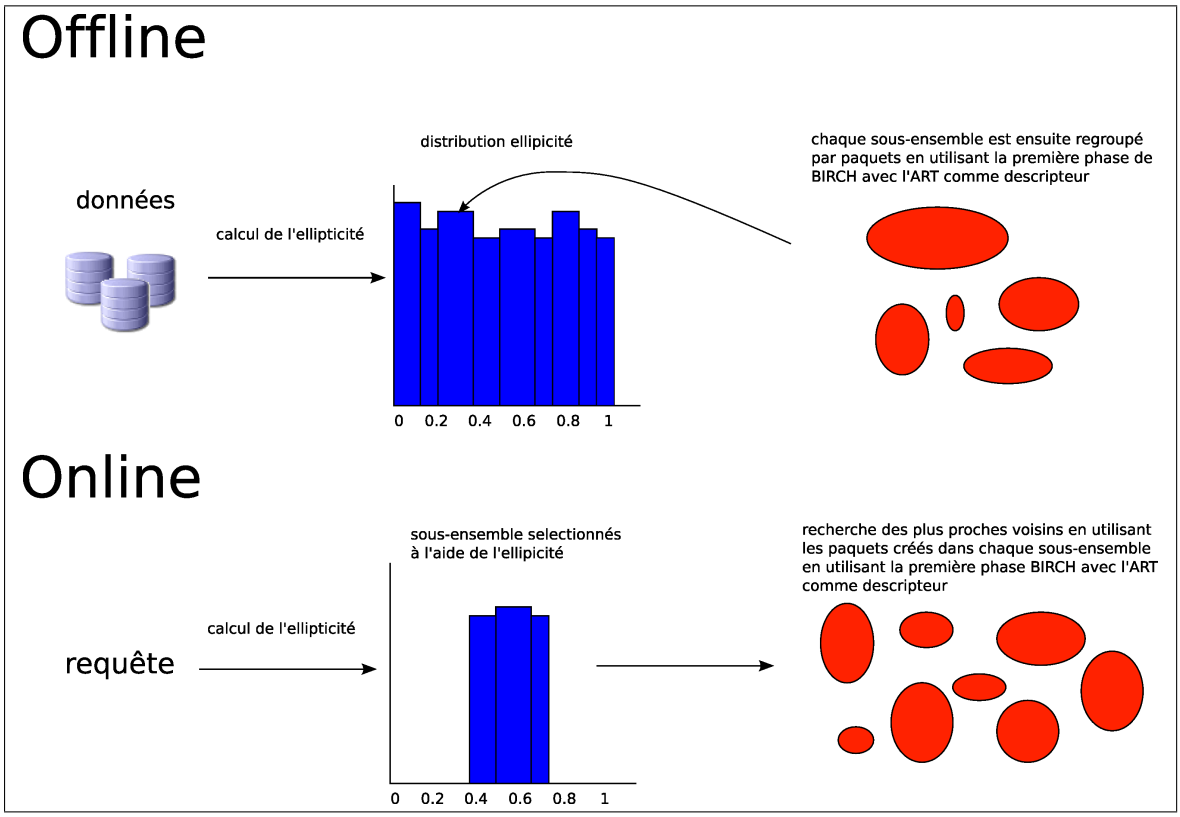


FIG. 5.13 – Méthode proposée.

On peut alors imaginer étendre ce procédé à plusieurs descripteurs. Chaque descripteur nous conduit à ne retenir qu'un ensemble restreint de données. En prenant l'intersection de ces ensembles, nous éliminons encore plus de possibilités.

Ainsi, il est possible d'utiliser plusieurs descripteurs, cependant ils doivent répondre à certains critères. Le descripteur doit être à valeur réelle, donc un vecteur à une dimension et ce afin de pouvoir définir un seuil qui ait un sens. Ce que nous entendons par là, c'est que l'on doit pouvoir comprendre l'effet de ce seuil. Ainsi, si fixer un seuil pour l'ellipticité a un réel sens, fixer un seuil à une des dimensions du descripteur ART n'en a pas, et serait totalement arbitraire. D'une manière générale, quand un descripteur possède plusieurs dimensions, il est souvent difficile de pouvoir interpréter chacune de ses dimensions, et donc de fixer avec certitude un seuil au-delà duquel nous pouvons dire que deux symboles ne sont plus semblables.

Pour notre étude nous avons choisi les descripteurs suivants :

- D_1 : l'ellipticité
- D_2 : le nombre de points de jonctions que l'on trouve à partir du squelette
- D_3 : l'aire contenue dans le rectangle englobant le symbole

D_1 et D_2 sont a priori des descripteurs robustes, tandis que D_3 lui n'est pas invariant à l'échelle. Il est là pour montrer qu'il est bien sur possible d'intégrer des descripteurs qui

ne soient pas invariants à des critères que l'utilisateur peut, pour des raisons personnelles, ne pas vouloir considérer.

5.3.2 Evaluations des paramètres

Les seuils Regardons dans un premier temps les seuils que nous pouvons fixer pour les descripteurs $D1$, $D2$ et $D3$. Pour chacun de ces descripteurs, nous allons regarder l'évolution du taux de filtrage et du taux de bons symboles écartés en fonction d'un seuil. Soit q la requête, et x un des éléments de la base, les seuils pour les différents descripteurs s_{D_1}, s_{D_2} et s_{D3} sont définis par :

$$|D_1(q) - D_1(x)| \leq s_{D_1}$$

$$|D_2(q) - D_2(x)| \leq s_{D_2}$$

$$max(|D_3(q)/D_3(x)|, |D_3(x)/D_3(q)|) \leq s_{D_3}$$

Nous avons utilisé une distance euclidienne pour les deux premiers descripteurs, et une mesure basée sur un rapport pour le descripteur utilisant l'aire. Ces distances nous semblaient assez bien adaptées à ce que nous voulions faire, mais il est bien entendu possible d'en utiliser d'autres. Les taux de filtrages et de bons symboles écartés pour les différents descripteurs en fonction de leur seuil sont illustrés dans les figures 5.14 à 5.19. Il en ressort que l'ellipticité permet d'avoir un bon filtrage, puisqu'avec un seuil de 0.2, on obtient un taux de filtrage de 90% et ceci sans éliminer de bons candidats. Le descripteur D_2, basé sur le nombre de points de jonction, permet quant à lui d'obtenir pour un seuil de 5 un taux de filtrage de plus de 50%. Ce taux de filtrage faible peut s'expliquer par le fait notre méthode génère de nombreux candidats qui n'ont pas de points de jonctions, ce qui affecte donc les recherches quand nous essayons justement de retrouver des symboles qui n'ont pas de points de jonctions. Quant au dernier descripteur D_3 basé sur l'aire, il permet d'avoir un taux de filtrage assez élevé (de l'ordre de 80%), même en considérant les symboles qui font 2 fois la taille du symbole modèle, ce qui dans la plupart des cas permet de ne pas éliminer de bons symboles candidats.

Nous avons donc vu trois descripteurs différents qui permettent d'avoir des taux de filtrages élevés tout en préservant les bons symboles. Pour la suite, nous considérerons, conformement à notre étude, que $s_{D_1} = 0.2$, $s_{D_2} = 5$ et $s_{D_3} = 4$. L'idée maintenant est de les combiner, afin de ne garder que l'intersection des ensembles qu'ils décrivent. La figure Fig 5.3.2 donne le taux de filtrage en combinant les différents descripteurs. Il en ressort qu'en combinant les trois descripteurs on atteint un taux de fitrage de 98%, alors que dans le meilleur des cas auparavant, on n'avait que 89% (ce qui correspond au taux de filtrage pour l'ellipticité avec un seuil à 0.2). Soit un gain de plus de 10 points. Cela confirme bien l'interêt de combiner les descripteurs et ceux-ci en particulier.

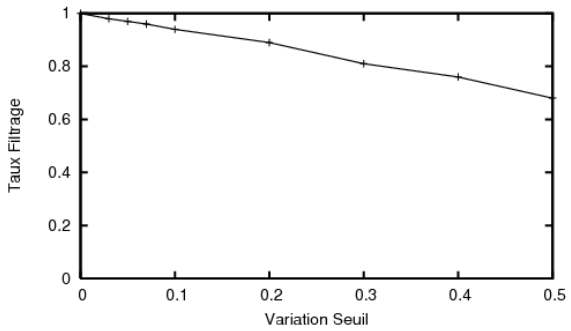


FIG. 5.14 – Taux de filtrage en utilisant le descripteur D_1 et en fonction du seuil s_{D_1}.

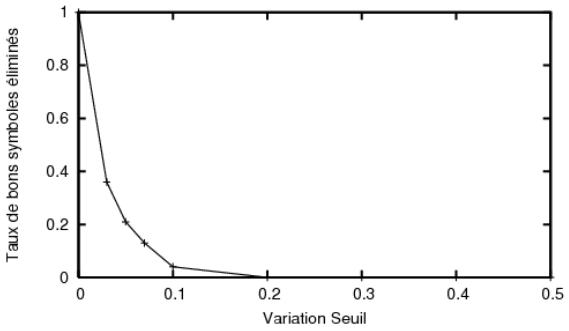


FIG. 5.15 – Taux de bons symboles éliminés en utilisant le descripteur D_1 et en fonction du seuil s_{D_1}.

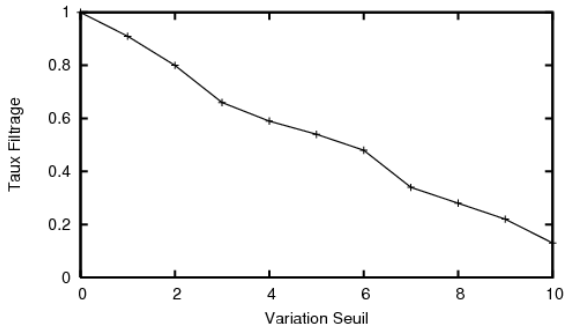


FIG. 5.16 – Taux de filtrage en utilisant le descripteur D_2 et en fonction du seuil s_{D_2}.

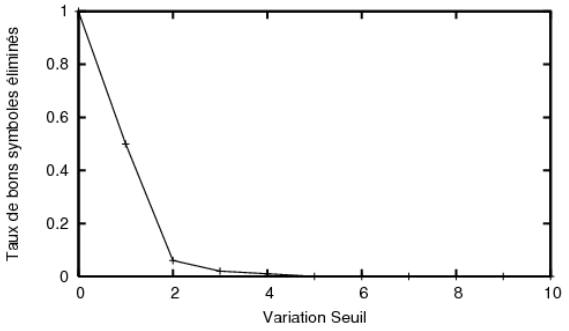


FIG. 5.17 – Taux de bons symboles éliminés en utilisant le descripteur D_2 et en fonction du seuil s_{D_2}.

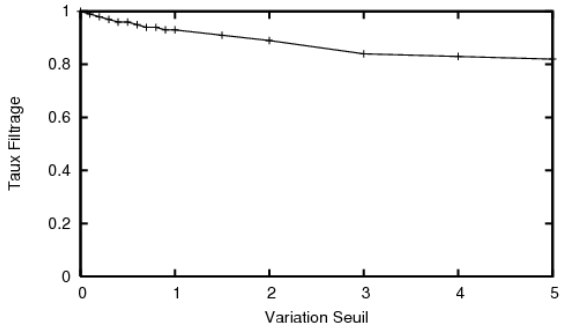


FIG. 5.18 – Taux de filtrage en utilisant le descripteur D_3 et en fonction du seuil s_{D_3}.

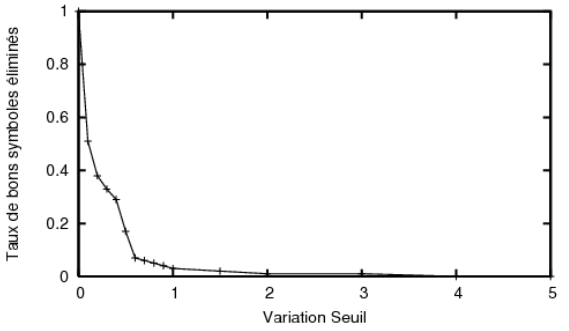


FIG. 5.19 – Taux de bons symboles éliminés en utilisant le descripteur D_3 et en fonction du seuil s_{D_3}.

Combinaison	Taux de filtrage
D_1,D_2	94%
D_1,D_3	90%
D_2,D_3	96%
D_1,D_2,D_3	98%

FIG. 5.20 – Taux de filtrage en fonction des différentes combinaisons entre descripteur.

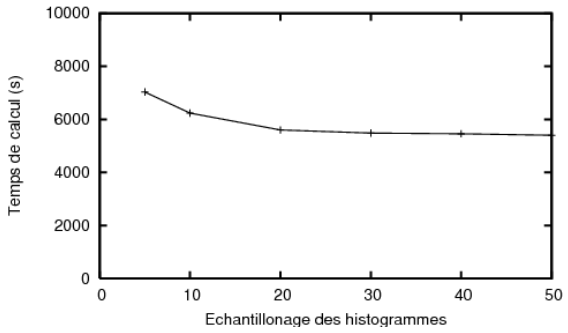


FIG. 5.21 – Temps de calcul (en s) pour des échantillonages différents des histogrammes.

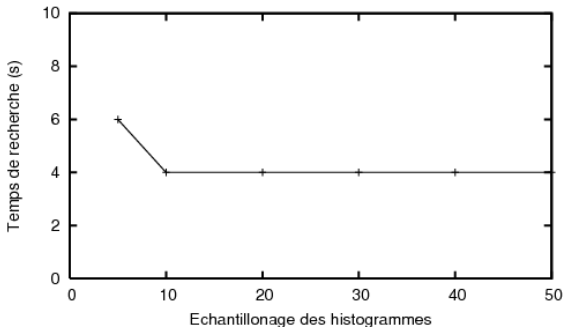


FIG. 5.22 – Temps de réponse (en s) pour des échantillonages différents des histogrammes.

Echantillonage des histogrammes L'échantillonage des histogrammes permet de regrouper en paquets les données selon leur descripteur. Ainsi, dans notre cas, nous aurons un histogramme pour le descripteur D_1, un autre pour D_2 et encore un autre pour D_3. Bien échantillonner l'histogramme est important. En effet, par la suite nous sélectionnerons les données selon cet échantillonage. Ne pas le diviser assez nous conduirait à sélectionner des paquets qui contiendront trop de données ne répondant pas au seuil fixé. En revanche, trop le diviser nous conduirait certes à ne considérer que des données pertinentes, mais il faut alors garder en tête que l'on appliquera pour chaque paquet la méthode de regroupement par paquets que nous avons présenté plus tôt (voir 5.2). Ainsi, nous aurons des paquets à l'intérieur des paquets. On conçoit alors que multiplier ces paquets risque de faire chuter les performances en terme de temps de recherche des plus proches voisins. Les figures 5.21 et 5.22 donnent le temps de calcul et de recherche nécéssaire en fonction de l'échantillonage choisi. Les temps de recherche sont assez stables, ils tournent autour de 4s, à part pour un échantillonnage de 5 où il est de 6s. Ceci s'explique par le fait que, dans ce cas on choisit les paquets beaucoup plus grossièrement et donc on récupère des éléments qui auraient du être éliminés. Les temps de calcul eux diminuent quand l'échantillonage augmente, ce qui s'explique par le fait que le regroupement par paquets à l'intérieur des histogrammes prend moins de temps quand la taille du paquet initial est petite (c'est le principe du diviser pour conquérir).

5.3.3 Résultats

La figure Fig 5.23, montre les courbes de précision/rappel en faisant une recherche des 100 plus proches voisins en utilisant l'ART pour différentes valeurs d'échantillonage des histogrammes. La courbe de référence est la courbe qui a été obtenue dans la partie précédente avec une simple recherche séquentielle des plus proches voisins.

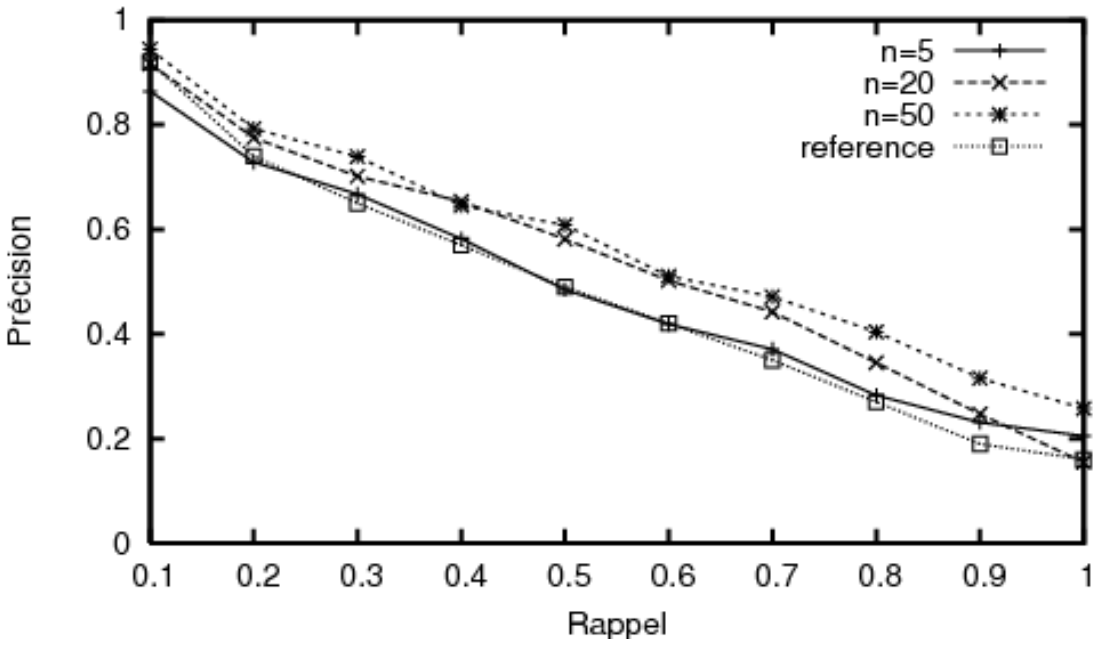


FIG. 5.23 – Courbes de rappel/précision pour différentes valeurs d'échantillonage des histogrammes, à comparer avec la courbe de rappel/précision de référence qui est celle qui a été obtenue en utilisant une recherche des plus proches voisins séquentiels.

Il en ressort donc que nous avons amélioré la méthode sur de nombreux points. Certes le temps de calcul hors-ligne a augmenté puisqu'il est passé de 50min à 1h30, mais ce n'est en soit pas prohibitif. Par contre non seulement nous avons amélioré le temps de recherche qui est descendu à 4s (alors que, pour rappel, la méthode séquentielle prenait 18s, soit un gain de 400 %), mais aussi la courbe de précision/rappel s'est également améliorée. Ceci étant dû au fait que le filtrage que nous effectuons filtre les symboles candidats à 98 % (dans le cas de l'utilisation combinée de D_1, D_2 et D_3) ce qui permet d'augmenter les performances du descripteur. De plus alors que le taux de symboles manqués en considérant les 500 plus proches voisins auparavant était de 56%, il n'est plus ici que de 9%. La figure 5.24 montre une partie des résultats obtenues avec notre interface.

5.4 Conclusion

Dans ce chapitre nous avons proposé au départ une amélioration d'une technique permettant de regrouper les données par paquets. Ce regroupement devant aidé à accélérer

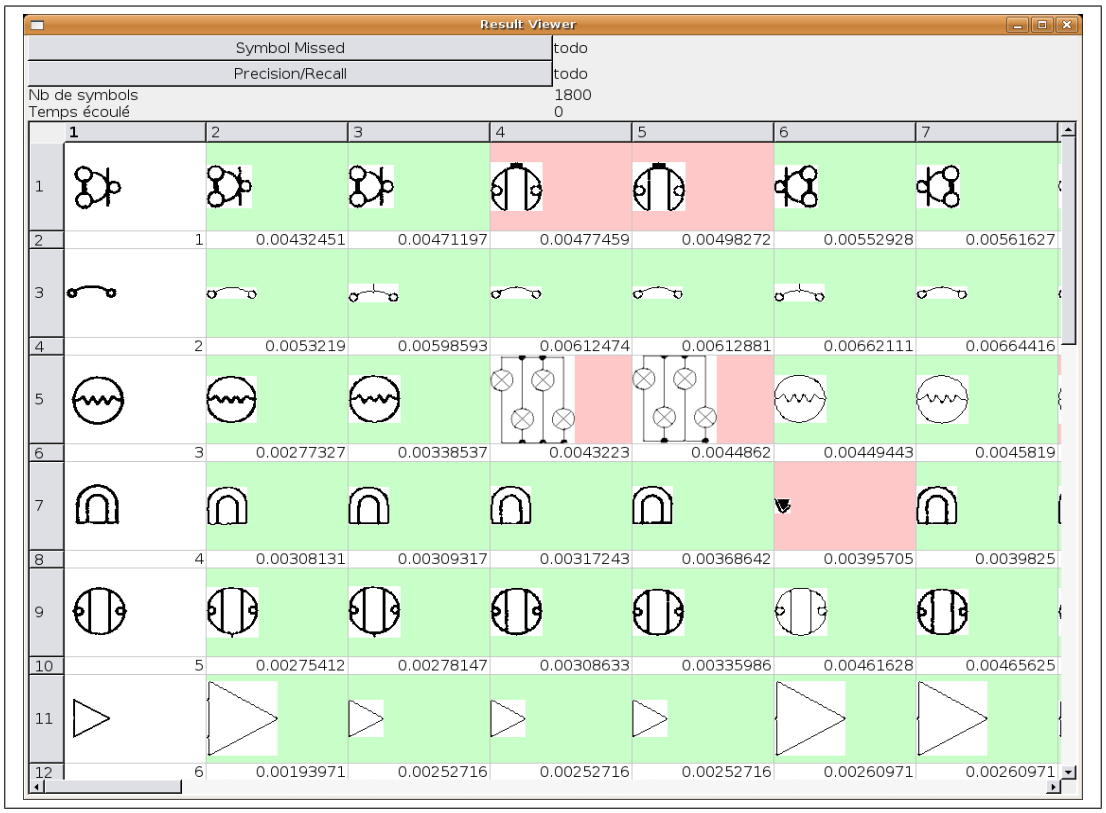


FIG. 5.24 – Fenêtre montrant les résultats de la reconnaissance.

la recherche des plus proches voisins. Nous avons montré que ce regroupement par paquets permettait de diviser le temps de recherche par deux.

Puis nous avons proposé une autre amélioration qui consiste à regrouper les données selon leur vecteur caractéristique pour différent descripteurs simples, ce qui permet une fois que l'on soumet la requête de ne considérer que les paquets les plus pertinents. Nous avons alors montré que ce filtrage permet de diviser ce temps de réponse par deux tout en améliorant les performances.

Nous avons donc au final proposé une méthode qui est 4 fois plus rapide que la recherche séquentielle des plus proches voisins tout en améliorant les performances de presque 10% du point de vue de la courbe de précision/rappel, et qui a ramené le taux de symboles manqués de 56% à 9%.

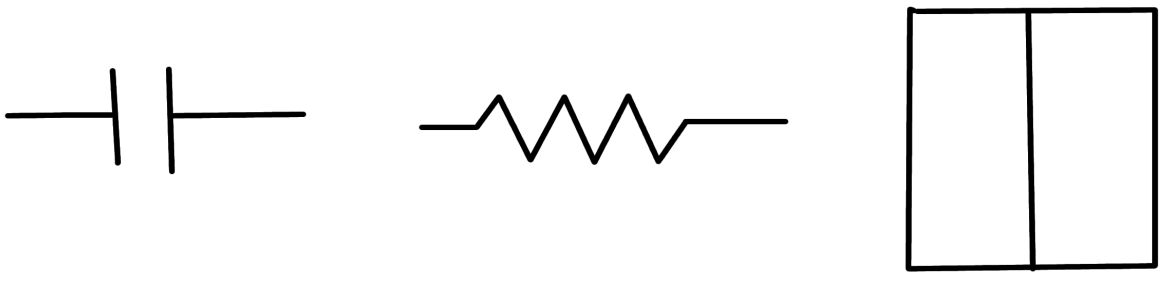
Conclusion

Nous avons vu que la reconnaissance de symboles dans des documents graphiques peut se diviser en différentes phase : représentation, description et classification. Nous avons abordé dans ce travail ces différentes phases.

La phase de description a été abordé sous l'angle d'une étude comparative de performance de quatre descripteurs (Radon, ART, Zernike et Yang). Nous avons mené ces expérimentations en utilisant différentes base de données et différents critères d'évaluation. Cette étude nous a alors amené à considérer l'ART comme étant le descripteur à utiliser pour notre usage.

Nous ensuite abordé la phase de représentation, en mettant en avant une méthode originale de segmentation des symboles dans des documents graphiques. Cette méthode repose sur une décomposition du document en chaînes de points que nous avons ensuite fusionner entre elles selon un certain critère qui tient compte de l'aspect général des symboles. Les résultats obtenus semblent prometteurs, même si dans un premier temps nous avons pu voir que 56% des symboles étaient manqués. De plus certains types de symboles ne peuvent être correctement segmentés pour l'instant.

En premier lieu, les symboles qui sont composés de chaînes de point déconnectés, comme par exemple les capacités (voir Fig 1(a)) ne sont pas pris en charge. Mais afin de surmonter ce problème, on peut, avant de construire le dendrogramme, enrichir le graphe, en parcourant le document et trouvant des chaînes de points qui bien qu'elles ne soient



(a) (b) (c)

Fig. 1 – Symboles et cas non traités avec la méthode proposée.

pas connectées, sont suffisament proches l'une de l'autre. Cela revient alors à rajouter des arcs dans le graphe. Puis le reste de la méthode est identique.

Le deuxième cas non traité est celui des symboles qui, bien que non isolés dans le document, ne présentent pas de points de jonctions, comme par exemple une résistance (voir Fig 1(b)). Pour pouvoir traiter ce cas, il faudrait changer la manière dont on construit le graphe, et ne plus le construire en se basant sur les points de jonctions et les points terminaux issus d'une squelettisation, mais en se basant sur une approximation polygonale par exemple. Ainsi, dans le cas de la résistance, celle-ci serait alors composée de différents segments qui seraient les nœuds dans le graphe. En gardant le reste de la méthode, leur détection serait possible.

Un dernier cas qui n'est pas traité avec cette méthode est celui où l'on aurait deux symboles côte à côte qui auraient en commun une chaîne de points (voir Fig 1(c)). Dans ce cas, la méthode ne pourra pas segmenter correctement les deux symboles à la fois. Dans le meilleur des cas, elle n'en segmentera qu'un seul. Ceci étant du au fait que l'on fusionne successivement les chaînes de points. Une fois de plus ce problème pourrait être contourner en changeant la manière dont on construit le graphe. Au lieu de considérer les chaînes de points, il faudrait alors considérer les boucles primaires qui composent le document.

Ces différents cas nous montrent bien l'impossibilité de traiter tous les documents avec une méthode unique. Certains nécessitent un graphe basé sur des points de jonctions, d'autres nécessitent un graphe basé sur une approximation polygônale, tandis que d'autres nécéssiteraient un graphe basé sur les régions. La solution est peut-être alors d'appliquer ces trois méthodes différentes successivement. Cependant, la méthode qui permet de fusionner les nœuds du graphe semble être générique puisque rien ne paraît empêcher son utilisation une fois le bon type de graphe construit. Tout au plus faudrait-il considérer d'autres critères d'agrégation.

Le principal problème de cette méthode est le nombre assez conséquent de symboles candidats qu'elle retourne (200000 symboles candidats pour 100 documents). Nous avons dû mettre en œuvre une méthode d'indexation afin de pouvoir trouver rapidement les plus proches voisins par rapport à une requête donnée mais il pourrait être possible de filtrer en amont ces candidats, ce qui présente toutefois des difficultés si on ne dispose pas ou si on ne veut pas injecter de connaissances a priori. Cependant un filtrage en injectant un minimum de connaissance pourrait être envisageable en exploitant la structure hiérarchique des ces candidats via le dendrogramme. Des tests ont été menés, mais ils n'ont pour l'instant pas été concluants.

La phase classification a donc été abordée sous l'angle de la recherche des plus proches voisins. A cause du grand nombre de symboles à considérer, la recherche séquentielle prennait 18s. Nous avons alors montré qu'avec un simple regroupent par paquets, ce temps descend à 8s. Puis nous avons mis en œuvre une méthode originale de filtrage se basant sur des descripteurs simples qui permet de filtrer près de 98% des symboles, ramenant le temps de recherche à 4s, soit un gain de plus de 400% par rapport à la recherche séquentielle. Et qui de plus améliore les résultats puisqu'on arrive alors à un taux de symboles manqués de 9% quand il était de 56% auparavant.

La méthode d'indexation semble assez bien appropriée puisqu'elle permet des taux de filtrage élevés pour des temps de réponse relativement bas. Si la base venait à augmenter on pourrait alors imaginer l'utilisation d'une recherche approximatives des plus proches voisins afin de réduire si nécessaire le temps de réponse, avec en contre partie des performances moindres sur la précision de la recherche.

Des recherches doivent encore être menées, en particulier sur d'autres type de documents. En effet, même si nous avons soutenu l'idée que cette méthode était générique, nous n'avons pas pu mener des expérimentations que sur des documents électronique en raison du manque de base de documents disponible. Cela montre aussi l'effort qui doit être fait afin de disposer de base de données accessibles aux chercheurs afin de mener les expérimentations et les études comparatives nécessaires.

Ce thème de recherche est donc loin d'être épuisé et de nombreux points restent à résoudre. Nous espérons cependant y avoir contribué et que les travaux que nous avons menés ([TWZ04], [ZT06a], [ZR06] et [ZT06b]) permettront de faire avancer les autres travaux à venir.

Annexe A

Calcul rapide de la transformée de Radon

Afin de réduire la complexité, on peut calculer la transformée de Radon avec la méthode proposée par [Bra98]. L'idée consiste à calculer de manière récursive les lignes qui balayent l'image suivant un certain θ. On comprend bien qu'à partir d'un même point, la ligne traversant ce point pour $\theta = 0$ a beaucoup de points en commun avec la ligne traversant ce même point pour un θ proche de 0. La méthode utilise donc des sommes partielles afin de réduire la complexité à $O(N^2 logN)$. Prenons, par exemple, une image de taille 8x8 (Fig A.1).

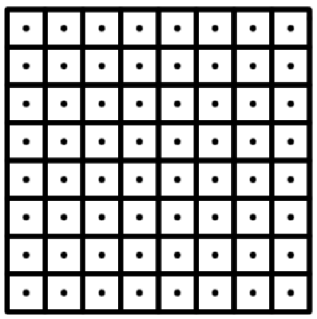


Fig. A.1 – Image Originale de taille 8x8

Calculer la transformée de Radon revient à calculer pour un ensemble de θ, la somme des pixels le long d'une droite d'angle θ et à des distances variables. Les figures A.2 en donnent un exemple. Pour chaque angle, on doit ici calculer 8 sommes correspondant aux 8 θ qui décrivent l'intervalle [0,π/4] (la méthode s'adapte facilement pour décrire les autres intervalles). Ainsi on a $\theta_i = atan(i/8)$ avec $i \in [0, 7]$.

On voit visuellement que l'on calcule souvent la même chose. En effet, la somme de deux pixels côte à côte est utilisée dans les 4 cas de figures présentés. En stockant ces sommes et en ne les calculant qu'une seule fois, on va pouvoir accélérer l'algorithme.

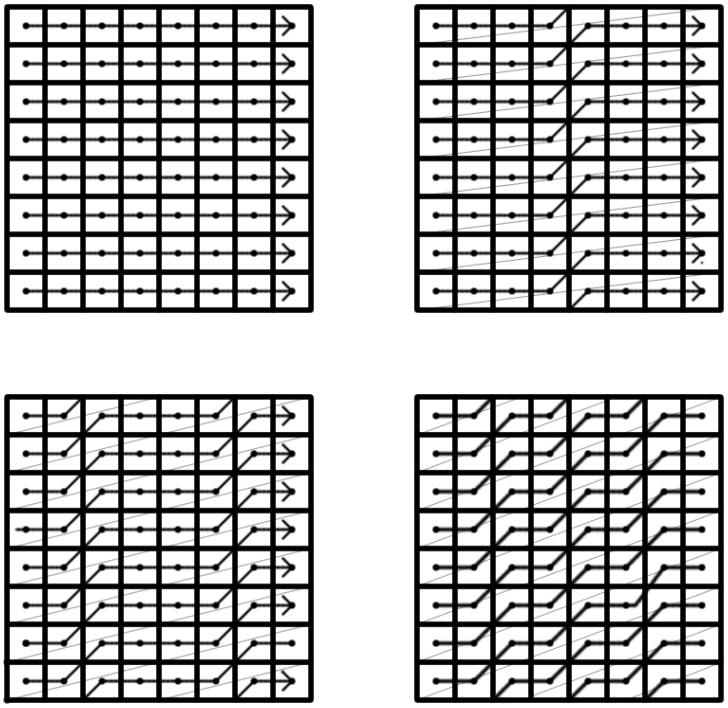


FIG. A.2 – Les sommes à calculer pour θ_i, i de 0 à 4

L'idée est donc de partir de l'image de départ et de calculer la somme de termes deux à deux suivant deux directions différentes (horizontale et diagonale) conformément à la figure A.3.

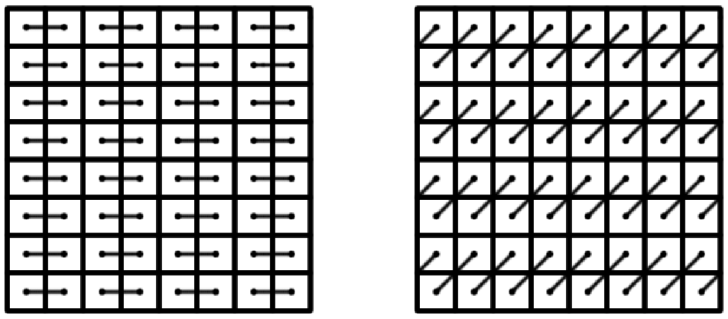


FIG. A.3 – Calcul sur la matrice suivant deux directions, il en résulte Md et Mh

Il va en résulter deux matrices de taille 4x8, auxquelles on va appliquer le même procédé (Fig A.4).

On a alors 4 matrices de taille 2x8 (Fig A.6). Ainsi la matrice Mdd et Mdh représente les sommes qui sont la figure A.5. Puis en réitérant une dernière fois, on obtient au final 8 matrices de taille 1x8.

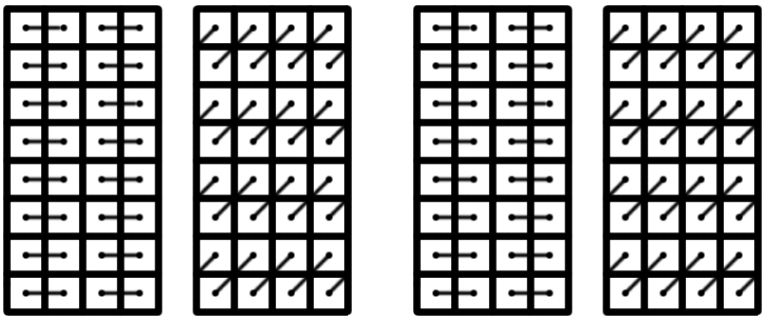


FIG. A.4 – Matrices issues du procédé : Mdd, Mdh, Mhd et Mhh

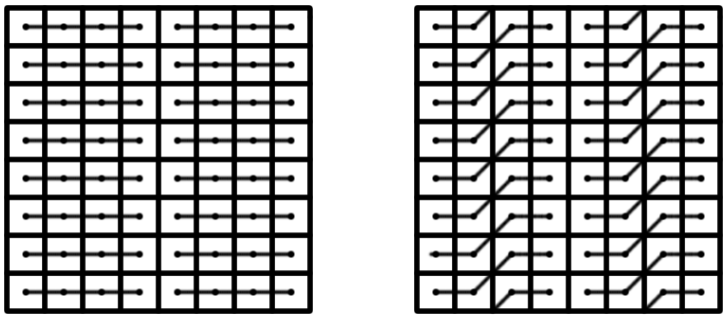


FIG. A.5 – Sommes issues de Mdd et Mdh

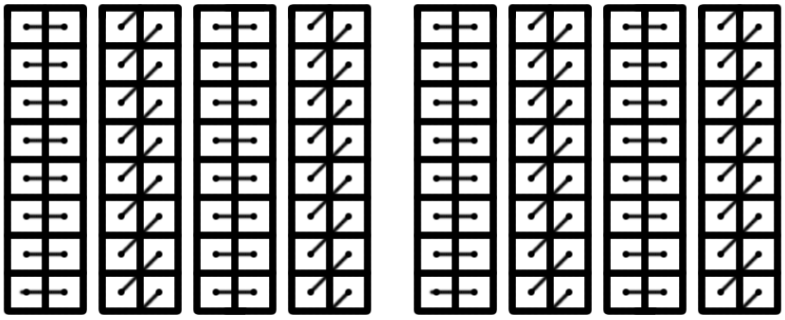


FIG. A.6 – Matrices issues du procédé : Mddd, Mdhd, Mdhd,Mdhh, Mhdd, Mhdh, Mhhd et Mhhh

On peut alors remarquer que les sommes correspondant à θ_1 correspondent à Mddd. Celle à θ_2 à Mddh, et ainsi de suite. On a alors pu calculer de manière rapide la transformée de Radon. L'exemple donné ici calcule la transformée de Radon sur l'intervalle $[0, \Pi/4]$. Pour obtenir les autres intervalles, il suffit de choisir des directions différentes.

Afin d'obtenir la signature que nous avons défini plus haut, il faut alors, comme précédemment, calculer la somme carrée des éléments d'une même colonne. Mais, si nous faisons uniquement cela, la signature va perdre toutes ses propriétés intéressantes. En effet les données issues de cette méthode ne sont pas échantillonnées de manière uniforme que ce soit en ρ ou en θ.

Echantillonnage en ρ. Tout d'abord en ce qui concerne l'échantillonnage en ρ, il va falloir diviser par $cos(\theta)$ chaque donné. Ceci correspond à l'espacement entre des lignes successives (à θ fixé). Ainsi pour $\theta = 0$, l'espacement est de 1, alors que pour $\theta = \pi/4$, l'espacement est de $1/\sqrt{(2)}$.

Echantillonnage en θ. Pour l'échantillonnage en θ, la méthode utilise $\theta = atan(a/N - 1)$ avec $a = 0, 1, 2, ..., N - 1$ avec N étant la taille de l'image. Il nous faut donc interpoler les données, afin d'avoir un pas régulier. Nous avons donc interpolé les données afin de ne prendre les valeurs qu'avec θ décrivant l'intervalle [0,180] degré avec un pas de 1. La signature garde alors toutes ses propriétés.

Annexe B

Présentation de l'interface

Nous présentons ici une interface qui a été développée afin de pouvoir évaluer notre méthode. Cette interface répond à un cahier des charges spécifiques que nous présentons ici. L'objectif principal est de fournir à l'utilisateur une interface qui lui permette de traiter en bloc un ensemble de documents. On distingue alors trois types de traitements que peuvent subir ces documents :

- Un document peut être simplement transformé en un ou plusieurs autres documents, comme dans le cas d'une binarisation.
- Un document peut être segmenté, c'est-à-dire que l'on va isoler des zones susceptibles de contenir des symboles.
- On peut faire une reconnaissance de symboles sur le résultat d'une segmentation.

Cette interface repose en grande partie sur la plate-forme logiciel QgarSofware développée par l'équipe Qgar . Elle reprend l'interface graphique existante et l'étend pour nos propres besoins. L'interface dans son ensemble représente environ 110000 lignes de code.

B.1 Les nœuds

Cette interface est basée sur la notion de nœuds. Un nœud est une sorte de boîte noire qui a une ou plusieurs données en entrées, et qui renvoie une ou plusieurs données en sortie. Les nœuds peuvent, par ailleurs, se chaîner les uns les autres. En accord avec nos propres besoins, nous avons introduit trois types de nœuds :

- un *processing node* qui transforme un document en un ou plusieurs documents
- un *segmentor node* qui segmente les documents
- un *comparator node* qui va reconnaître les symboles segmentés

Nous allons maintenant voir plus particulièrement chaque type de nœuds.

B.1.1 Processing Node

Le *processing node* transforme un document en un ou plusieurs documents. Ainsi pour une binarisation, on ne possède qu'une image en sortie, mais pour une séparation texte/graphique, on aura une image représentant la partie reconnue comme du texte, une autre image représentant la partie reconnue comme du graphique et même une autre image regroupant ce que la méthode n'a pas plus classer comme du texte ou du graphique. Ainsi le *processing node* possède les caractéristiques suivantes (Fig B.1) :

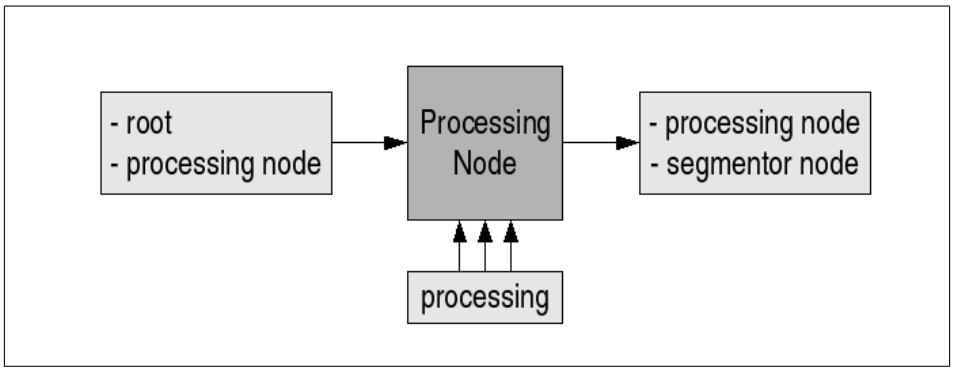


FIG. B.1 – Les entrées/sorties possibles, ainsi que les paramètres d'un processing node.

- En entrée, il peut soit être au début de l'arbre de traitements, soit succéder à un autre *processing node*.
- En sortie, il peut être suivi soit d'un autre *processing node*, soit d'un *segmentor node*.
- De plus, il prend en paramètre ce que l'on appelle un *processing* qui représente le traitement qui va être effectué sur les images en entrée.

B.1.2 Segmentor Node

Le *segmentor node* segmente les documents. Il a les caractéristiques suivantes (Fig B.2) :

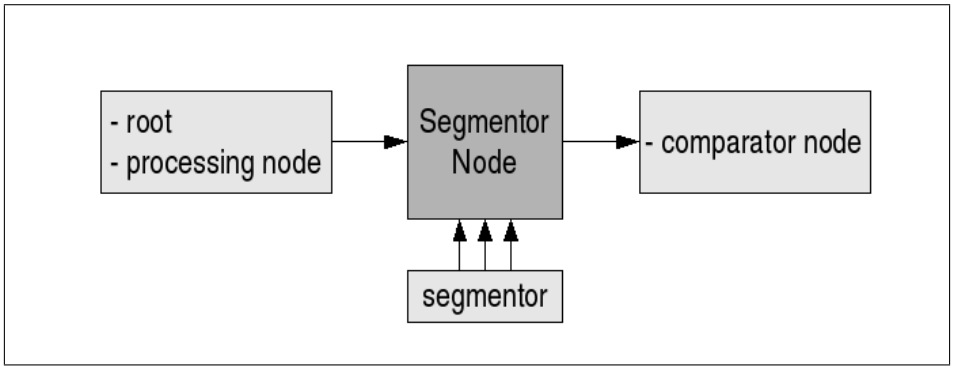


FIG. B.2 – Les entrées/sorties possibles, ainsi que les paramètres d'un segmentor node.

– En entrée, il peut soit être au début de l'arbre de traitements, soit succéder à un autre *processing node*.
– En sortie, il ne peut être suivi que d'un *comparator node*.
– De plus, il prend en paramètre ce que l'on appelle un *segmentor* qui représente la segmentation qui va être effectuée sur les images en entrée.

B.1.3 Comparator Node

Le *comparator node* va quant à lui permettre de faire une reconnaissance de symboles dans les documents. Il a les caractéristiques suivantes (Fig B.3) : :

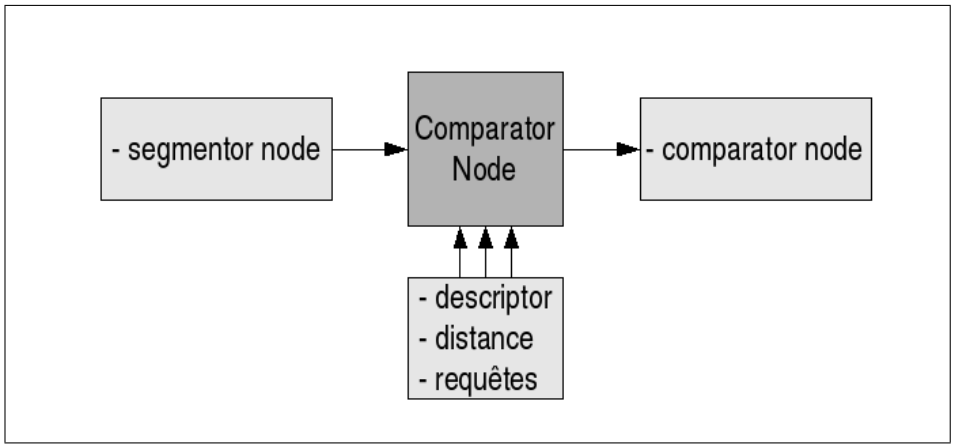


FIG. B.3 – Les entrées/sorties possibles, ainsi que les paramètres d'un comparator node.

– En entrée, il ne peut recevoir que les résultats d'une segmentation, c'est pourquoi il possède forcément en entrée un *segmentor node*.
– En sortie, il ne peut être suivi que d'un *comparator node*, si on veut par exemple utiliser différents descripteurs les uns à la suite des autres.
– Il prend différents paramètres comme le *descriptor* qui représente le descripteur à utiliser, la *distance* à utiliser, les symboles à trouver, et la méthode de classification.

B.1.4 Les différents paramètres

Les différents paramètres comme le *processing*, le *segmentor*, le *descriptor*, et la *distance*, sont tous identifiés par un couple (nom méthode, vecteur<paramètre>). De plus tout a été conçu afin de pouvoir intégrer facilement tout nouveau *processing*, *segmentor*, *descriptor* ou *distance*.

B.2 Les données et leur intégration

Les données dans l'interface s'articule autour de deux classes : QGDocumentDataBase qui est relative aux documents à traiter, et une seconde classe QGSymbolDataBase qui

est relative aux symboles à trouver.

B.2.1 QGDocument DataBase

Cette classe regroupe d'une part l'ensemble des documents à traiter et d'autre part les traitements qu'ils subissent.

B.2.1.1 L'ensemble des documents.

L'utilisateur alimente la base de documents soit en spécifiant le chemin d'un document, soit en spécifiant un répertoire contenant les documents à traiter.

B.2.1.2 Les traitements.

Les traitements se composent de *processing node*, de *segmentor node* et de *comparator node*, accessibles depuis des boîtes de dialogue (Fig B.4). Une fenêtre (Fig B.5) permet d'avoir un aperçu de tous les traitements qui ont été effectués, ainsi que de tous les documents qui sont traités. Il est alors possible d'afficher un document après un traitement particulier. Sur la Fig B.5, on peut voir que différents traitements ont été effectués. D'une part, à partir des documents originaux, on a fait une séparation texte/graphique qui a pour résultat trois images différentes représentant les composantes textuelles, les composantes graphiques et les composantes indéterminées. La composante graphique a ensuite subi une segmentation du nom de "segment graph". Les documents originaux ont par ailleurs aussi subi une binarisation, ainsi qu'une segmentation du nom de "segment graph". En sélectionnant le traitement représentant les composantes graphiques, l'image du document sélectionné correspondant s'affiche (Fig B.6).

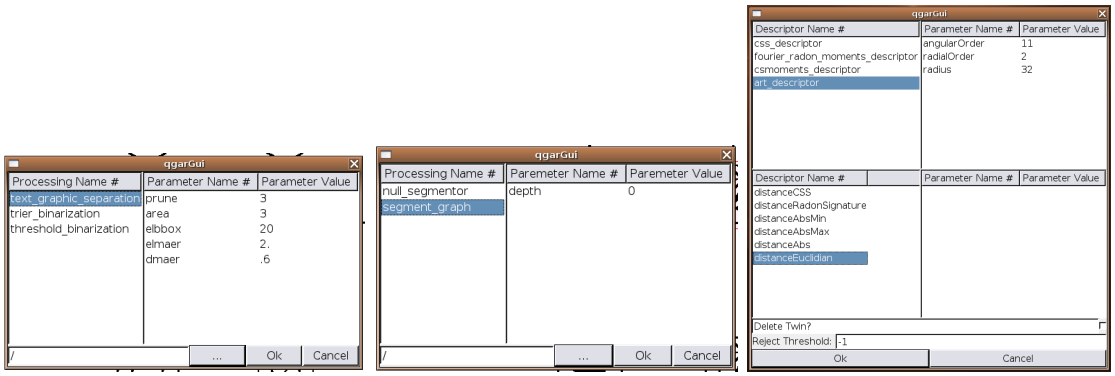


FIG. B.4 – Les boîtes de dialogue pour le *processing node*, de *segmentor node* et de *comparator node*

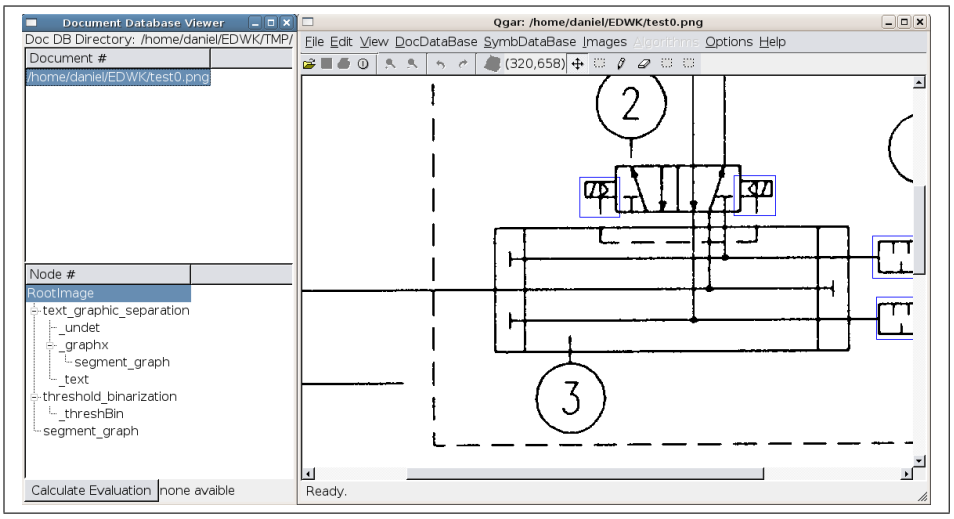


FIG. B.5 – Pas de traitements sélectionnés.

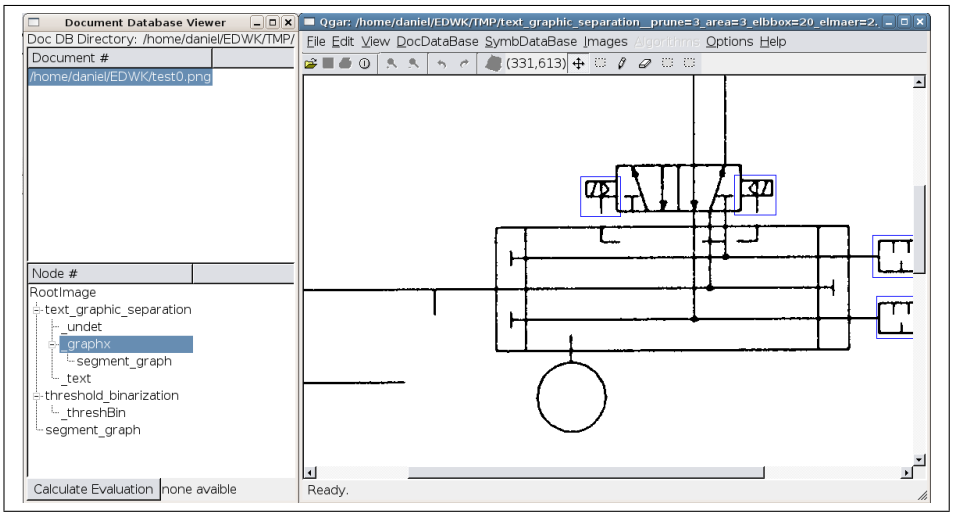


FIG. B.6 – Sélection de la composante graphique de la séparation texte/image (graphx est sélectionné dans la fenêtre de gauche).

B.2.1.3 La vérité terrain.

De plus, à chaque document, est associée une vérité terrain si elle a été définie. La vérité terrain est un ensemble de couples (rectangle englobant, étiquette). L'interface permet donc à l'utilisateur de délimiter une zone de l'image qui contient un symbole et de l'étiqueter (Fig B.7). De plus, il est possible d'accéder à un tableau récapitulatif qui montre toutes les zones contenant des symboles, ce qui permet de vérifier qu'il n'y a pas eu d'erreurs d'étiquetage (Fig B.8) et permet de les corriger le cas échéant.

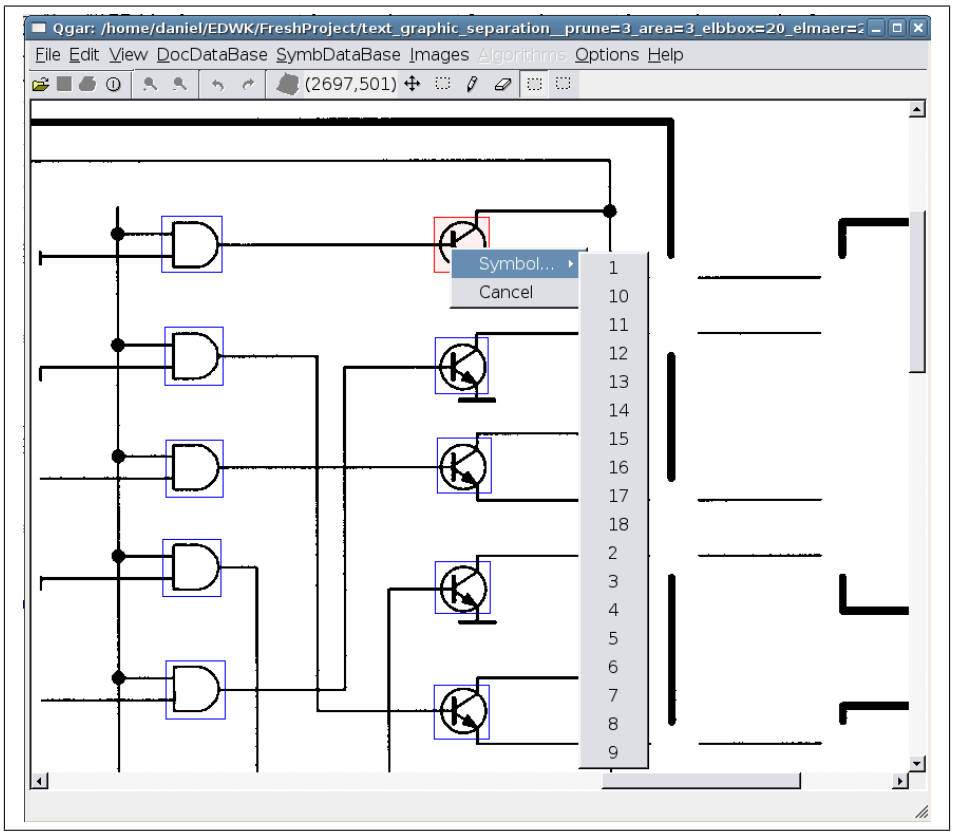


FIG. B.7 – Sélection d'une zone (en rouge), puis attribution d'une étiquette (ici les symboles ont pour étiquette un nombre allant de 1 à 18).

B.2.2 QGSymbolDataBase

Cette classe contient la liste des symboles à trouver dans les documents. Elle peut être alimentée de deux manières différentes : soit l'utilisateur importe des images qu'il considère comme des symboles, soit il peut directement sélectionner, dans les documents segmentés (Fig B.9), des zones qu'il considère comme contenant un symbole à rechercher. Une fenêtre permet également de contrôler les symboles modèles que l'on considère en permettant également de modifier leur étiquette (Fig B.10).

B.2.3 Les résultats

Une fois que l'on dispose de documents qui ont été segmentés, ainsi que d'une base de symboles à trouver, on peut alors lancer la recherche des symboles. Les résultats s'affichent dans une fenêtre à part (Fig B.11). La première colonne montre les symboles recherchés, et les colonnes suivantes montrent les symboles trouvés par pertinence décroissante. La distance au modèle est par ailleurs indiquée. Les symboles retrouvés correspondant à une vérité terrain se voient attribuer un fond vert (le fond le plus clair sur l'image), alors que ceux qui ne correspondent à aucune vérité terrain se voient attribuer un fond rouge (le fond le plus foncé sur l'image). Cette fenêtre permet également de connaître le nombre

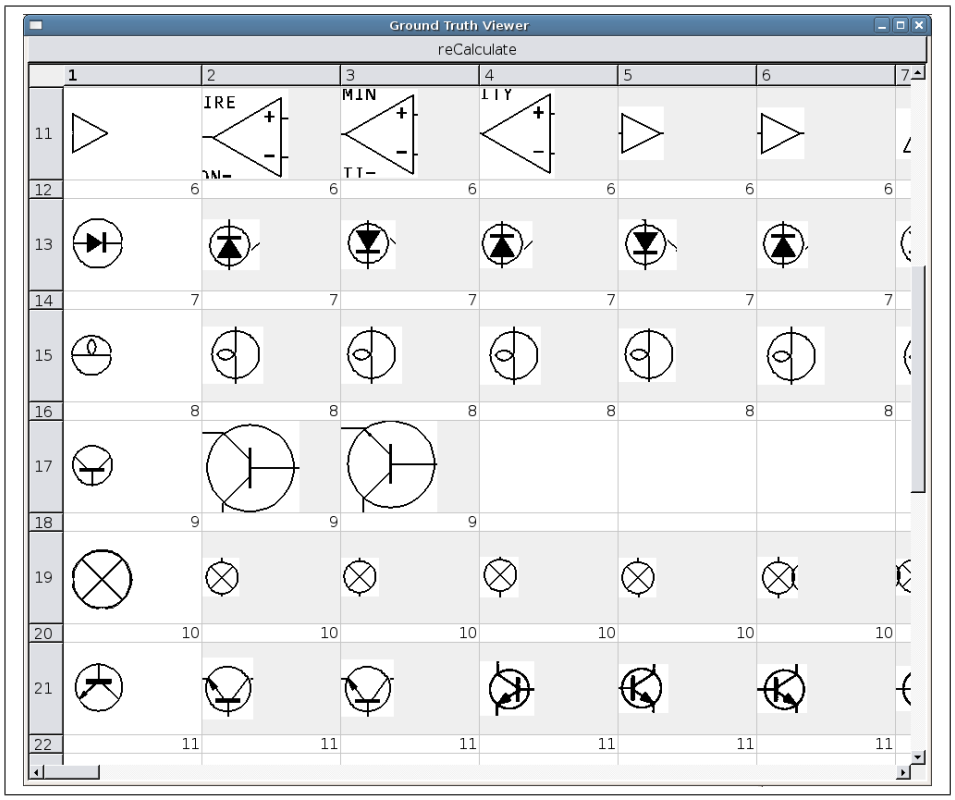


FIG. B.8 – Vue d'ensemble des symboles étiquetés par l'utilisateur.

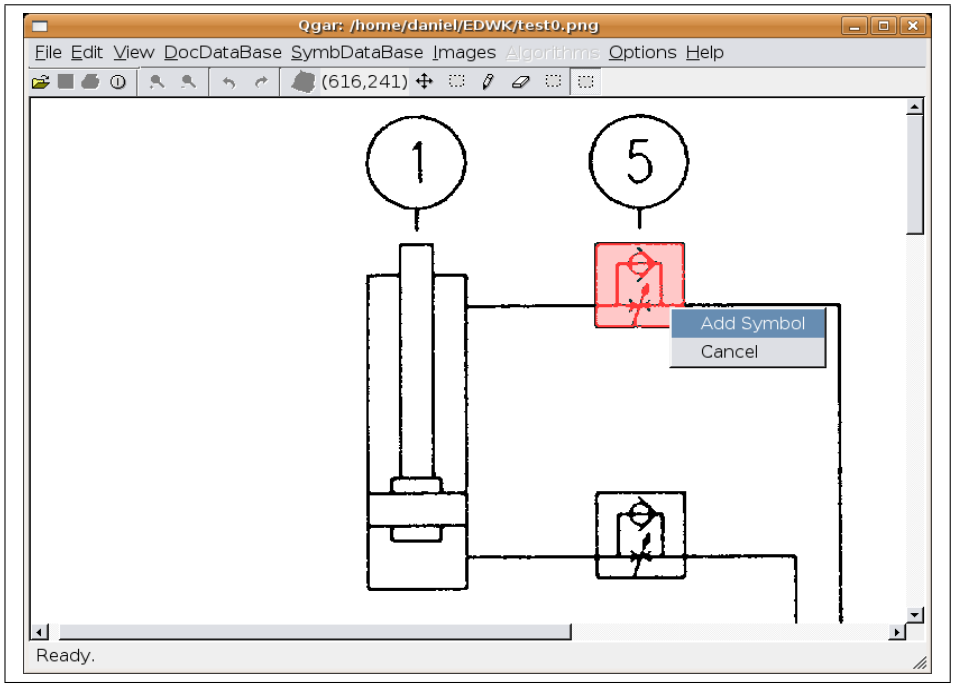


FIG. B.9 – Sélection d'un symbole à partir d'une segmentation.

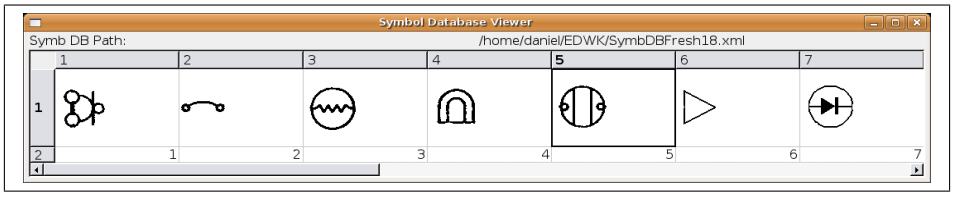


FIG. B.10 – Fenêtre récapitulant les symboles que l'on considère.

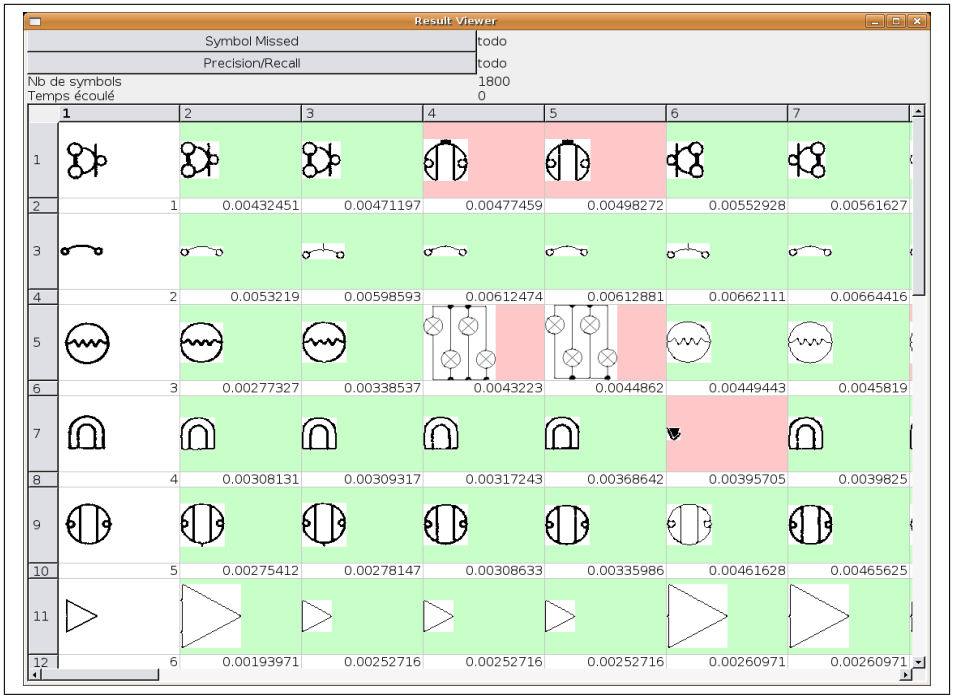


FIG. B.11 – Fenêtre montrant les résultats de la reconnaissance.

de symboles qui n'ont pas été reconnus (en accord avec la vérité terrain), ainsi que de calculer la courbe représentant la précision en ordonnée, et le rappel en abcisse.

Mais, la fonction particulièrement intéressante est celle qui permet de cliquer sur un des symboles retrouvés, et qui a pour effet de localiser le symbole dans son contexte. C'est ce qu'illustre la figure B.12, en cliquant sur le symbole situé dans la colonne deux en haut de l'écran, la fenêtre du bas s'est mise automatiquement sur le document correspondant et montre par un fond bleu le symbole dans son contexte.

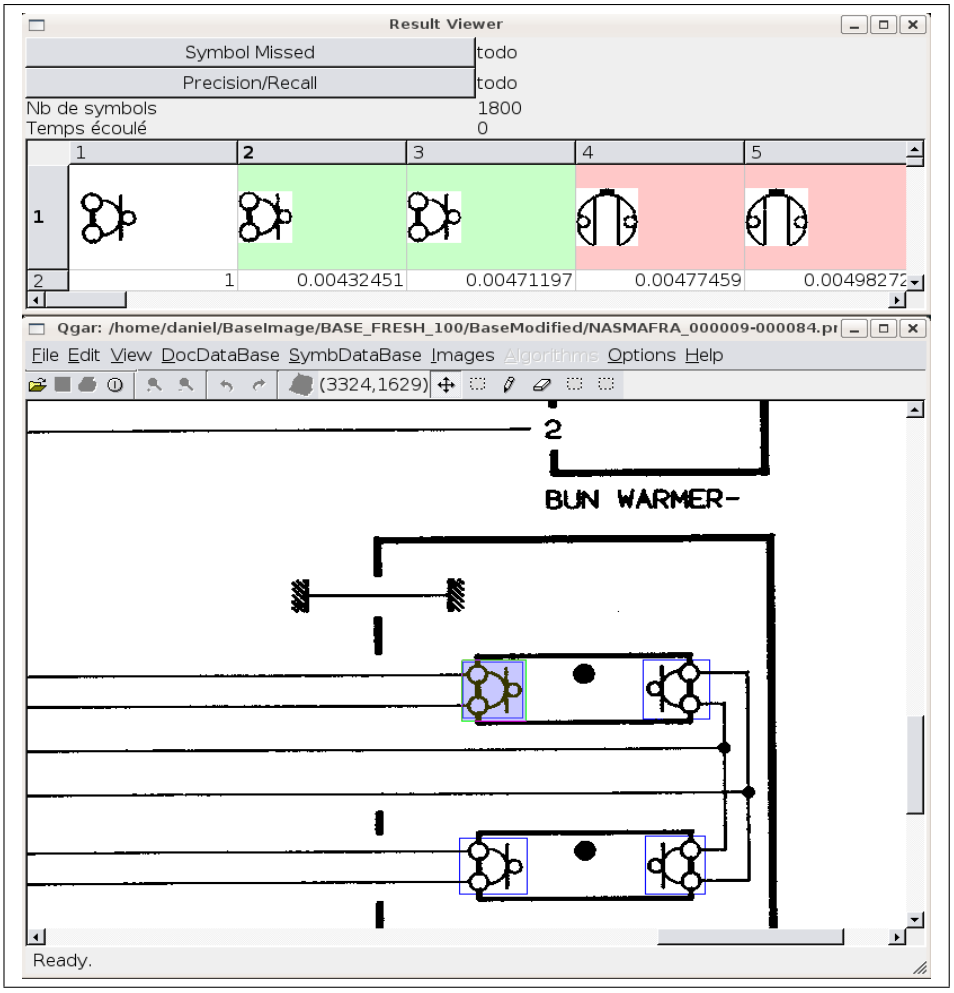


FIG. B.12 – Après avoir cliqué sur le symbole situé en colonne 2, on resitue ce symbole dans son document et on le met en avant par un fond bleu.

Bibliographie

[ABKJ99] Michael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and J.Sander. OPTICS : ordering points to identify the clustering structure. In *SIGMOD '99 : Proceedings of the 1999 ACM SIGMOD, International Conference on Management of Data*, pages 49–60, 1999.

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD '98 : Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 94–105, New York, NY, USA, 1998. ACM Press.

[Ant91] D. Antoine. *Interprétation des plans cadastraux français à partir d'un modèle.* Thèse de doctorat, Institut National Polytechnique de Lorraine, Vandœuvre-lès-Nancy, 1991.

[AOCG01] S. Adam, J.-M. Ogier, C. Cariou, and J. Gardes. A Scale and Rotation Parameters Estimator : Application to Technical Document Interpretation. In *Proceedings of 4th IAPR International Workshop on Graphics Recognition, Kingston, Ontario (Canada)*, pages 377–380, September 2001.

[BBK98] S. Berchtold, C. Bohm, and H.P. Kriegel. The pyramid-technique : Towards breaking the curse of dimensionality. In *Proceedings of the ACM SIGMOD International Conference on Management Data*, pages 142–153, 1998.

[BCB+92] L. Boatto, V. Consorti, M. Del Buono, S. Di Zenzo, V. Eramo, A. Esposito, F. Melcarne, M. Meucci, A. Morelli, M. Mosciatti, S. Scarci, and M. Tucci. An Interpretation System for Land Register Maps. *IEEE COMPUTER Magazine*, 25(7) :25–33, July 1992.

[Bel61] R. Bellman. On the Approximation of Curves by Line Segments using Dynamic Programming. *Communications of the ACM*, 4(6) :284, 1961.

[Ber73] A. T. Berztiss. A backtrack procedure for isomorphism of directed graphs. *Journal of the ACM*, 20(3) :365–377, 1973.

[Ber02] P. Berkhin. Survey of Clustering Data Mining Techniques. Technical report, Acrrue Software, San Jose, CA, 2002.

[Ber04] S.A. Berrani. *Recherche approximative de plus proches voisins avec contrôle probabiliste de la précision ; application à la recherche d'images par le contenu.* thèse, Université de Rennes 1, 2004.

[Bez81] J. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum, New York, 1981.

[BGRS99] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When Is "Nearest Neighbor" Meaningful ? *Lecture Notes in Computer Science*, 1540 :217–235, 1999.

[BH67] G. Ball and D. Hall. A clustering technique for summarizing multivariate data. *Behavior Science*, 12(2) :153–155, 1967.

[BKK96] S. Berchtold, D.A. Keim, and H.P. Kriegel. The X-Tree : An Index Structure for High-Dimensional Data. In *Proceedings of the 22nd International Conference on Very Large Databases*, pages 28–39, 1996.

[BKS90] N. Beckmann, H.P. Kriegel, and R. Schneider B. Seeger. The R*-tree : An efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.

[Bob01] M. Bober. Mpeg-7 visual shape descriptor. *IEEE Trans. Circuits Syst. Video Technol*, 1(6), 2001.

[Bol98] Daniel Boley. Principal Direction Divisive Partitioning. *Data Mining and Knowledge Discovery*, 2(4) :325–344, 1998.

[Boz83] H. Bozdogan. Determining the Number of Component Clusters in the Standard Multi-variate Normal Mixture Model Using Model-Selection Criteria. Technical report, University of Illinois, ARO Contract DAAG29-820K- 0155, Quantitative Methods Department, Chicago, Illinois 60680, 1983.

[Boz94] H. Bozdogan. Mixture-Model Cluster Analysis Using Model Selection Criteria and a New Informational Measure of Complexity. In *Proceedings of the First US/Japan Conference on the Frontiers of Statistical Modeling : An Informational Approach*, pages 69–113, Dordrecht,the Netherlands, 1994. Kluwer Academic Publishers.

[Bra95] R. N. Bracewell. *Two-Dimensional Imaging*. Prentice Hall Signal Processing Series, New York, 1995.

[Bra98] M. L. Brady. A Fast Discrete Approximation Algorithm For The Radon Transform. *SIAM Journal on Computing*, 27(1) :107–119, February 1998.

[BSA91] S. O. Belkasim, M. Shridar, and M. Ahmadi. Pattern Recognition with Moment Invariants : A Comparative Study and New Results. *Pattern Recognition*, 24 :1117–1138, 1991.

[Bun82] H. Bunke. Attributed Programmed Graph and Application to Diagram Interpretation. *IEEE Transactions on PAMI*, 6(4) :574–582, 1982.

[CFCW01] T. ChiuT, D. Fang, J. Chen, and Y. Wang. A robust and scalable clustering algorithm for mixed type attributes in large database environments. In *Proceedings of the 7th ACM SIGKDD, 2001*, volume 1, pages 263–268, 2001.

[CFSV04] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3) :265–298, 2004.

[CG92] G. Celeux and G. Govaert. A classification EM algorithm for clustering and two stochastic versions. *Computational Statistics and Data Analysis,*, 14 :315–332, 1992.

[CH74] T. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in statistics*, 3(1) :1–27, 1974.

[CH87] G. L. Cash and M. Hatamian. Optical Character Recognition by the Method of Moments. *Computer Vision, Graphics and Image Processing*, 39 :291–310, 1987.

[Cho65] N. Chomsky. *Aspects of the Theory of Syntax*. MIT Press, 1965.

[CLW+96] L.-H. Chen, H.-Y. Liao, J.-Y. Wang, K.-C. Fan, and C.-C. Hsieh. An Interpretation System for Cadastral Maps. In *Proceedings of the 13th International Conference on Pattern Recognition, Vienna (Austria)*, volume 3, pages 711–715, August 1996.

[Coü96] B. Coüasnon. *Segmentation et reconnaissance de documents guidées par la connaissance a priori : application aux partitions musicales*. Thèse de doctorat, Université de Rennes I, January 1996.

[CPZ97] P. Ciacca, M. Patella, and P. Zezula. M-Tree : An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Databases*, pages 426–435, 1997.

[CRM03] C. Chong, P. Raveendran, and R. Mukudan. A comparative analysis of algorithms for fast computation of zernike moments. *Pattern Recognition*, 36 :731–742, 2003.

[CV00] L. P. Cordella and M. Vento. Symbol recognition in documents : a collection of techniques ? *International Journal on Document Analysis and Recognition*, 3(2) :73–88, December 2000.

[DBM77] S. A. Dudani, K. J. Bredding, and R. M. McGhee. Aircraft Identifiation by Moment Invariants. *IEEE Transactions on Computers*, 26 :39–45, 1977.

[Dea83] S. R. Deans. *Applications of the Radon Transform*. Wiley Interscience Publications, New York, 1983.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification, Second Edition*. Wiley-InterScience, 2000.

[dHtKG94] J. E. den Hartog, T. K. ten Kate, and J. J. Gerbrands. An Alternative to Vectorization : Decomposition of Graphics into Primitives. In *Proceedings of Third Symposium on Document Analysis and Information Retrieval, Las Vegas*, April 1994.

[EKSX96] M. Ester, H-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[Fed71] J. Feder. Plex Languages. *Information Science*, 3 :225–241, 1971.

[FK88] L. A. Fletcher and R. Kasturi. A robust algorithm for text string separation from mixed text/graphics images. *IEEE Transactions on PAMI*, 10(6) :910–918, 1988.

[FK99] Hichem Frigui and Raghu Krishnapuram. A Robust Competitive Clustering Algorithm With Applications in Computer Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5) :450–465, 1999.

[FL03] Ana L.N. Fred and Leitao Leitao. A new cluster isolation criterion based on dissimilarity increments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8) :944–958, 2003.

[Fu74] K. S. Fu. *Syntactic Methods in Pattern Recognition*, volume 112 of *Mathematics in Science and Engineering*. Academic Press, New York, 1974.

[Fu83] K. S. Fu. A Step Towards Unification of Syntactic and Statistical Pattern Recognition. *IEEE Transactions on PAMI*, 5(2) :200–205, 1983.

[GCB04] N. Grira, M. Crucianu, and N. Boujemaa. Unsupervise and Semi-supervised Clustering : a Brief Survey. Technical report, INRIA Rocquencourt, Le Chesnay, France, 2004.

[Gra72] G.H. Granlud. Fourier preprocessing for handprint character recognition. *IEEE Trans. Comput C-21*, pages 195–201, 1972.

[GRG+99] Venkatesh Ganti, Raghu Ramakrishnan, Johannes Gehrke, Allison L. Powell, and James C. French. Clustering Large Datasets in Arbitrary Metric Spaces. In *ICDE*, pages 502–511, 1999.

[GRS98] S. Guha, R. Rastogi, and K. Shim. CURE : an efficient clustering algorithm for large databases. In *Proceedings fo the 1998 ACM SIGMOD international conference on Management of data*, volume 9, pages 73–84, 1998.

[Gut84] A. Gutman. R-trees : A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[Hab93] A. H. Habacha. A New System for the Analysis of Schematic Diagrams. In *Proceedings of 2nd International Conference on Document Analysis and Recognition, Tsukuba (Japan)*, pages 369–372, 1993.

[Hen98] A. Henrich. The lsd^h-tree : An access structure for feature vectors. In *Proceedings of the 12th International Conference on Data Engineering*, pages 362–369, 1998.

[HK98] Alexander Hinneburg and Daniel A. Keim. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. In *Knowledge Discovery and Data Mining*, pages 58–65, 1998.

[HN04] H. Hse and A. R. Newton. Sketched Symbol Recognition using Zernike Moments. In *Proceedings of the 17th International Conference on Pattern Recognition, Cambridge (UK)*, August 2004.

[Hol75] J.H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[HS00] Erez Hartuv and Ron Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4–6) :175–181, 2000.

[Hu62] M. K. Hu. Visual pattern recognition by moment invariants. *IRE Trans. on Information Theory*, 8 :179–187, 1962.

[JD88] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1988.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering : a review. *ACM Computing Surveys*, 31(3) :264–323, September 1999.

[KG82] F. P. Kuhl and C. R. Giardina. Elliptic Fourier Features of Closed Contours. *Computer Vision, Graphics and Image Processing*, 18 :236–258, 1982.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220 :671–680, 1983.

[KH90] A. Khotanzad and Y. H. Hong. Invariant Image Recognition by Zernike Moments. *IEEE Transactions on PAMI*, 12(5) :489–497, 1990.

[KHK99] George Karypis, Eui-Hong (Sam) Han, and Vipin Kumar. Chameleon : Hierarchical Clustering Using Dynamic Modeling. *Computer*, 32(8) :68–75, 1999.

[KK93] R. Krishnapuram and J.M. Keller. A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems*, 1(2) :98–110, 1993.

[KK99] W.-Y. Kim and Y.-S. Kim. A new region-based shape descriptor. In *TR 15-01, Pisa (Italy)*, December 1999.

[KR90] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data : An Introduction to Cluster Analysis*. John Wiley, 1990.

[KRCO90] R. Kasturi, R. Raman, C. Chennubhotla, and L. O'Gorman. Document Image Analysis : An Overview of Techniques for Graphics Recognition. In *Preproceedings of IAPR Workshop on Syntactic and Structural Pattern Recognition, Murray Hill, NJ (USA)*, pages 192–230, 1990.

[KS97] N. Katayama and S. Satoh. The sr-tree : An index structure for high-dimensional nearset neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on Management Data*, pages 36–380, 1997.

[KS02] C. Kan and M.D. Srinath. Invariant Character Recognition with Zernike and Orthogonal Fourier-Mellin Moments. *Pattern Recognition*, 35 :143–154, 2002.

[LJF94] K.I. Lin, H.V. Jagadish, and C. Faloutsos. The tv-tree : An index structure for high-dimensional data. *VLDB journal*, 3(4) :517–542, 1994.

[LLKM97] J. Lladós, J. López-Krahe, and E. Martí. A System to Understand Hand-Drawn Floor Plans Using Subgraph Isomorphism and Hough Transform. *Machine Vision and Applications*, 10(3) :150–158, 1997.

[LMG+98] A. Lassaulzais, R. Mullot, J. Gardes, J. M. Ogier, and Y. Lecourtier. Segmentation d'infrastructures de réseau téléphonique. In *Actes du 1er Colloque International Francophone sur l'Écrit et le Document, Québec (Canada)*, pages 188–197, May 1998.

[LSM98] J. Lladós, G. Sánchez, and E. Martí. A String Based Method to Recognize Symbols and Structural Textures in Architectural Plans. In K. Tombre and A. K. Chhabra, editors, *Graphics Recognition—Algorithms and Systems*, volume 1389 of *Lecture Notes in Computer Science*, pages 91–103. Springer-Verlag, April 1998.

[LVSM01] J. Lladós, E. Valveny, G. Sánchez, and E. Martí. Symbol Recognition : Current Advances and Perspectives. In *Proceedings of 4th IAPR International Workshop on Graphics Recognition, Kingston, Ontario (Canada)*, pages 109–128, September 2001.

[LW67] G. Lance and W. Willimas. A general theory of classification sorting strategies. *Computer Journal*, 9 :373–386, 1967.

[MB96] B. T. Messmer and H. Bunke. Automatic Learning and Recognition of Graphical Symbols in Engineering Drawings. In R. Kasturi and K. Tombre, editors, *Graphics Recognition—Methods and Applications*, volume 1072 of *Lecture Notes in Computer Science*, pages 123–134. Springer-Verlag, May 1996.

[MC85] G. Milligan and M. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50 :159–179, 1985.

[McQ67] J. McQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[MH86] R. Mohr and T. C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28 :225–233, 1986.

[MK97] G. Mclachlan and T. Krishnan. *The EM algorithm and extensions*. Wiley, New York, NY, 1997.

[MKJ80] K. Mardia, J. Kent, and J.Bibby. *Multivariate Analysis*. Academic Press, 1980.

[MM92] F. Mokhatarian and A. K. Mackworth. A theory of multiscale, curvature-based shape representation for planar curves. *IEEE Transactions on PAMI*, 14(8) :789–805, 1992.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[Mur83] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *Computer Journal*, 26(4) :354–359, 1983.

[NGC99] H. Nagesh, S. Goil, and A. Choudhary. MAFIA : Efficient and scalable subspace clustering for very large data sets, 1999.

[NH02] Raymond T. Ng and Jiawei Han. CLARANS : A Method for Clustering Objects for Spatial Data Mining. *IEEE Trans. Knowl. Data Eng.*, 14(5) :1003–1016, 2002.

[NL90] V. Nagasamy and N. A. Langrana. Engineering Drawing Processing and Vectorization System. *Computer Vision, Graphics and Image Processing*, 49(3) :379–397, 1990.

[OKM+88] A. Okazaki, T. Kondo, K. Mori, S. Tsunekawa, and E. Kawamoto. An Automatic Circuit Diagram Reader with Loop-Structure-Based Symbol Recognition. *IEEE Transactions on PAMI*, 10(3) :331–341, 1988.

[PI97] M. Peura and J. Iivarinen. Efficiency of simple shape descriptors. In *3rd International Workshop on Visual Form*, pages 28–30, 1997.

[QR99] H. Qjidaa and L. Radouane. Robust line fitting in a noisy image by the method of moments. *IEEE Transactions on PAMI*, 21(11) :1216–1223, November 1999.

[RB79] Vijay V. Raghavan and Kim Birchard. A clustering strategy based on a formalism of the reproductive process in natural systems. In *SIGIR '79 : Proceedings of the 2nd annual international ACM SIGIR conference on Information storage and retrieval*, pages 10–22, New York, NY, USA, 1979. ACM Press.

[Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14 :464–471, 1978.

[RKD89] W. Raymond, R. Klein, and C. Dubes. Experiments in projection and clustering by simulated annealing. *Pattern Recognition*, 22(2) :213–220, 1989.

[Rob81] J.T. Robinson. The k-d-b-tree : A search structure for large multidimensional dynamic indexes. In *Proceedings of the ACM SIGMOD International Conference on Management Data*, pages 10–18, 1981.

[RPAK88] A. P. Reeves, R. J. Prokop, S. E. Andrews, and F. P. Kuhl. Three-Dimensional Shape Analysis Using Moments and Fourier Descriptors. *IEEE Transactions on PAMI*, 10(6) :937–842, 1988.

[RSB98] R.Weber, H.J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 194–205, 1998.

[Sch78] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6 :461–464, 1978.

[Sha69] A. C. Shaw. A Formal Picture Description Scheme as a Basis for Picture Processing Systems. *Information and Control*, 14(1) :9–52, 1969.

[Sne57] P. Sneath. The application of computers to taxonomy. *Journal of general microbiology*, 17 :201–226, 1957.

[Sor48] T. Sorensen. The application of computers to taxonomy. *Biologiske Skrifter*, 5 :1–34, 1948.

[SRF87] T.K. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-tree : A dynamic index for multi-dimensional objects. In *Proceedings of the 16th International Conference on Very Large data Bases*, pages 507–518, 1987.

[SS00] Roded Sharan and Ron Shamir. CLICK : Clustering Algorithm with Applications to Gene Expression Analysis. In *ISMB'00*, pages 307–316. AAAI Press, Menlo Park, CA, 2000.

[TB91] S.C.A. Thomopoulos and D.K. Bougoulias. DIGNET : a self-organizing neural network for automatic patternrecognition and classification. *Proceedings of the 30th IEEE Conference on Decision and Control*, 1 :853–858, 1991.

[Tea79] R. Teague. Image Analysis via the General Theory of Moments. *Journal of the Optical Society of America*, 70(8) :920–930, 1979.

[TFC04] R. Torres, A. Falcao, and L. Costa. A graph-based approach for multiscale shape analysis. *Pattern Recognition*, 37(6) :1163–1174, 2004.

[TJT96] Ø. Due Trier, A. K. Jain, and T. Taxt. Feature Extraction Methods for Character Recognition — A Survey. *Pattern Recognition*, 29(4), April 1996.

[TOD90] T. Taxt, J. B. Olafsdottir, and M. Daehlen. Recognition of Handwritten Symbols. *Pattern Recognition*, 23 :1155–1166, 1990.

[Tof96] P. Toft. *The Radon Transform – Theory and Implementation.* PhD thesis, Department of Mathematical Modelling, University of Denmark, 1996.

[TW02] S. Tabbone and L. Wendling. Technical Symbols Recognition Using the Two-dimensional Radon Transform. In *Proceedings of the 16th International Conference on Pattern Recognition, Québec (Canada)*, volume 2, pages 200–203, August 2002.

[TW03] S. Tabbone and L. Wendling. Binary shape normalization using the Radon transform. In *Proceedings of 11th International Conference on Discrete Geometry for Computer Imagery, Naples (Italy)*, volume 2886 of *Lecture Notes in Computer Science*, pages 184–193, November 2003.

[TWS06] S. Tabbone, L. Wendling, and J.-P. Salmon. A new shape descriptor defined on the Radon transform. *Computer Vision and Image Understanding*, 102(1) :42–51, April 2006.

[TWZ04] S. Tabbone, L. Wendling, and D. Zuwala. A hybrid approach to detect graphical symbols in documents. In S. Marinai and A. Dengel, editors, *Document Analysis Systems VI – Proceedings of 6th IAPR International Workshop on Document Analysis Systems, Florence (Italy)*, volume 3163 of *Lecture Notes in Computer Science*, pages 342–353, September 2004.

[Ull76] S. Ullman. Filling in the Gaps : The Shape of Subjective Contours and a Model for their Generation. *Biol. Cybernet*, 25 :1–6, 1976.

[Vap98] Vapnik. *Statistical Learning Theory.* Wiley, 1998.

[War63] J. H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58 :236–244, 1963.

[WF87] C. S. Wallace and P. R. Freeman. Estimation and inference by compact coding. *J. R. Statist. Soc. B*, 49(3) :240–265, 1987.

[WJ96] D. White and R. Jain. Similarity indexing with the ss-tree. In *Proceedings of the 12th International Conference on Data Engineering*, pages 516–523, 1996.

[XEKS98] Xiaowei Xu, Martin Ester, Hans-Peter Kriegel, and J. Sander. A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases. In *Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*, pages 324–331, 1998.

[XW05] R. Xu and D. Wunsch. Survey of Clustering Algorihtms. *IEEE Transactions on neural networks*, 16(3) :645–678, May 2005.

[Yam97] H. Yamada. Paper-Based Map Processing. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 19, pages 503–528. World Scientific, 1997.

[Yan05] S. Yang. Symbol Recognition via Statistical Integration of Pixel-Level Constraint Histograms : A New Descriptor. *IEEE Transactions on PAMI*, 27(2) :278–281, February 2005.

[YKTF94] N. Yoneda, K. Kise, S. Takamatsu, and K. Fukunaga. A Method of Understanding Conceptual Diagrams. In *Proceedings of IAPR Workshop on Machine Vision Applications, Kawasaki (Japan)*, pages 334–337, 1994.

[YSS97] Y. Yu, A. Samal, and S. C. Seth. A System for Recognizing a Large Class of Engineering Drawings. *IEEE Transactions on PAMI*, 19(8) :868–890, August 1997.

[YW04] Miin-Shen Yang and Kuo-Lung Wu. A similarity-based robust clustering method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(4) :434–448, 2004.

[YWB74] I. Young, J. E. Walker, and E. Bowie. An Analysis Technique for Biological Shape. *Information and Control*, 25 :357–370, 1974.

[Zah71] C.T. Zahn. Graph-Theorical methods for detecting and describing gestalt clusters. *IEEE Trans. Comput C-20*, pages 68–86, 1971.

[Zha02] Z. Zhang. A comparative study of fourier descriptors for shape representation and retrieval. In *fifth Asian Conference on Computer Vision*, volume 3, pages 646–651, 2002.

[ZL02] D.S Zhang and G. Lu. Generic Fourier descriptor for shape-based image retrieval. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, volume 1, pages 425–428, 2002.

[ZR72] C.T. Zahn and R.Z. Roskies. Fourier descriptors for planar closed curves. *IEEE Trans. Comput C-21*, pages 269–281, 1972.

[ZR06] D. Zuwala and J. Rendek. Browsing graphics without prior knowledge. In *Accepted for presentation at 18th International Conference on Pattern Recognition*, Hong Kong, China, August 2006.

[ZRL96] T. Zhang, R. Ramakrishnan, and L. Livny. BIRCH : an efficient data clustering method for vey large databases. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, volume 9, pages 103–114, 1996.

[ZS00] Jovisa Zunic and Natasa Sladoje. Efficiency of Characterizing Ellipses and Ellipsoids by Discrete Moments. *IEEE Transactions on PAMI*, 22(4) :407–414, 2000.

[ZT06a] D. Zuwala and S. Tabbone. A Method for Symbol Spotting in Graphical Documents. In H. Bunke and A. L. Spitz, editors, *Document Analysis Systems VII : Proceedings of 7th International Workshop on Document Analysis*

Systems, volume 3872 of *Lecture Notes in Computer Science*, pages 518–528, Nelson (New Zealand), February 2006.

[ZT06b] D. Zuwala and S. Tabbone. Une méthode de localisation de symboles sans connaissance a priori. In *CIFED 2006*, Fribourg, Swiss, September 2006.

INSTITUT NATIONAL
POLYTECHNIQUE
DE LORRAINE

AUTORISATION DE SOUTENANCE DE THESE
DU DOCTORAT DE L'INSTITUT NATIONAL
POLYTECHNIQUE DE LORRAINE

o0o

VU LES RAPPORTS ETABLIS PAR :
Monsieur Jean-Marc OGIER, Professeur, Université de la Rochelle, La Rochelle

Monsieur Laurent HEUTTE, Professeur, LITIS, UFR Sciences, Université de Rouen, Saint-Etienne-du-Rouvray

Le Président de l'Institut National Polytechnique de Lorraine, autorise :

Monsieur ZUWALA Daniel

NANCY BRABOIS
2, AVENUE DE LA
FORET-DE-HAYE
BOITE POSTALE 3
F - 54501
VANDŒUVRE CEDEX

à soutenir devant un jury de l'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE, une thèse intitulée :

"Reconnaissance de symboles sans connaissance a priori"

en vue de l'obtention du titre de :

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

Spécialité : « **Informatique** »

Fait à Vandoeuvre, le 23 octobre 2006
Le Président de l'I.N.P.L.,
L. SCHUFFENECKER

TEL. 33/03.83.59.59.59
FAX. 33/03.83.59.59.55