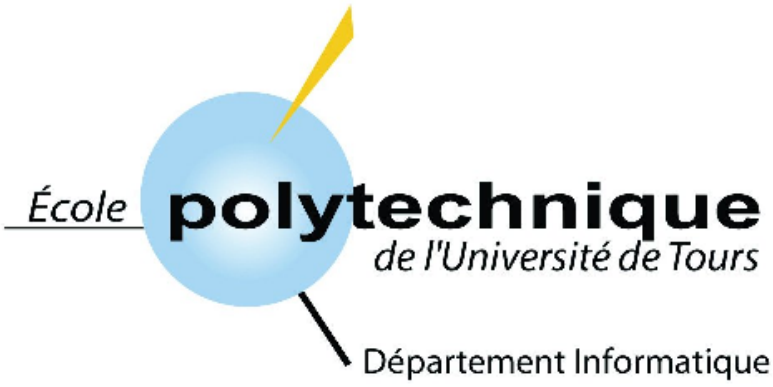
UNIVERSITE François Rabelais TOURS
École Polytechnique de Tours - Département Informatique
64, Avenue Jean Portalis
37200 TOURS



Rapport de stage de Fin d'études

⋆⋆⋆

Symbol Spotting : vers une reconnaissance intéractive des images de documents

Encadrant :
M. Jean-Yves RAMEL
LI Tours
Equipe RFAI

Etudiant :
M. Julien MICHOT
DI 5ème année
Promotion 2004-2007

Projet de Fin d'Etude

$\star\star\star$

Symbol Spotting : vers une reconnaissance intéractive des images de documents

Julien MICHOT

Septembre 2007

Remerciements

Ce stage de fin d'études s'est déroulé durant l'année scolaire 2006-2007, au sein de l'équipe RFAI du Laboratoire Informatique de l'Université de Tours, dirigée par M. Hubert Cardot. Je tiens à remercier toutes les personnes qui m'ont apporté leur soutien au cours de ce projet de fin d'études.

Je souhaite remercier particulièrement Jean-Yves Ramel, responsable adjoint de l'équipe RFAI, qui m'a encadré durant ce projet de fin d'études, pour ses conseils avisés, son encadrement ainsi que pour sa disponibilité.

Enfin, mes remerciements vont à l'ensemble de l'équipe RFAI, pour son accueil, son environnement de travail agréable et ses compétences.

Table des matières

1 Présentation du projet **3**
1.1 Cadre du projet 3
1.2 Présentation du problème 3
1.3 Description de l'environnement 4
1.4 Description des besoins à satisfaire 5
1.5 Contraintes 6
1.6 Plan de développement 7

2 Analyse et spécifications des besoins **9**
2.1 Introduction 9
2.2 Spécifications générales 9
2.2.1 Interaction Homme-Machine 10
2.2.2 Éléments de contenu et règles 11
2.2.3 Spécifications entrées-sorties 14
2.3 Liste des algorithmes nécessaires 15

3 Définitions et conception des éléments de la représentation d'images sous forme de graphe **18**
3.1 Modélisation sous forme de graphe orienté multivalué 18
3.2 Approximation des EdCs et détermination de l'axe d'inertie 20
3.3 Distance minimale entre deux EdCs 23
3.4 Calcul de l'angle inter-EdCs 24
3.5 Algorithme de construction du graphe de l'image 25
3.6 Calculs de nouveaux paramètres, mise à jour des informations 28

4 Modélisation UML des éléments du graphe **30**
4.1 Graph 30
4.2 Vertex (ou EdC) 32
4.3 Edge (Arc) 35

4.4 BoundingArea 38
4.5 Type d'élément de contenu 39
4.6 Type de relation 40

5 Algorithme de recherche de sous graphe minimal 42
5.1 Définition de la recherche 42
5.2 Interface de recherche de sous graphe 44
5.3 Algorithme 46
5.4 Heuristiques de choix des noeuds et arcs contraints à explorer 48
5.5 Propriétés de la méthode - résultats 49

6 Modules de gestion et de traitement 57
6.1 MainManager 57
6.2 DialogMain 58
6.3 Scénarii et règles d'action 59
6.3.1 La classe Regle 59
6.3.2 Interface de gestion des scenarii 60
6.3.3 Format XML des scenarii .sce 60
6.3.4 Creer une nouvelle regle 60
6.4 Structure des fichiers .sgxl 61

7 Le logiciel SymbSpot 62
7.1 Caractéristiques techniques 62
7.1.1 Librairie d'interfaces 62
7.1.2 Langage de programmation et gestionnaire de graphes/listes . . 63
7.1.3 Librairie de calculs graphiques 63
7.1.4 Tests unitaires et vérifications de l'efficacité du code 64
7.2 Fonctionnalités du logiciel 64
7.3 Évolutions et améliorations 68

Bibliographie 71

Introduction

Au cours de la scolarité à Polytech'Tours, nous sommes amenés à faire différents projets. Le plus conséquent d'entre eux est le projet de fin d'études (PFE). L'objectif pédagogique de ce projet est de mettre en application nos acquis afin de construire un projet complet, seul, comme si nous étions dans le monde du travail et que nous répondions à la demande d'un client.

Le sujet du PFE portait sur la recherche et la localisation de symboles présents dans des images de documents. La particularité des symboles est qu'ils devaient être de tout type, aussi bien des composants électroniques que des symboles de chimie ou plus abstrait encore, des zones d'images ou des paragraphes.

Dans ce document nous allons aborder les différents points de vue nécessaires à l'élaboration d'un logiciel. Le premier chapitre est consacré à la description du projet et de son environnement, puis nous aborderons ensuite les spécifications techniques du logiciel. Le troisième chapitre sera quant-à-lui organisé autour de la conception générale et détaillée de la modélisation d'images sous forme de graphes et du logiciel en lui-même. Enfin, nous terminerons par illustrer le logiciel que nous avons conçu, en détaillant les fonctionnalités implémentées au travers de quelques exemples.

Chapitre 1

Présentation du projet

Nous allons décrire dans ce chapitre les différents éléments constituant le cahier des charges du projet, nous présentons l'ensemble du projet dans son environnement ainsi que les démarches opérées lors de l'élaboration de celui-ci.

1.1 Cadre du projet

Le projet présenté dans ce document s'inscrit dans la continuité du parcours d'ingénieur en Informatique à l'école Polytech'Tours. Le projet de fin d'études modélise en quelque sorte les différentes étapes d'évolution d'un projet en entreprise, allant de l'étape de conception et d'analyse, de modélisation et de codage.

Ce projet de fin d'études a officiellement débuté en octobre 2006, principalement durant pendant du temps libre ; deux jours par semaines ont ensuite été alloués dès la mi-février afin de travailler entièrement sur le projet. La date de fin du projet est fixée à la fin du mois de septembre à la fin du stage de master 2 de recherche.

1.2 Présentation du problème

– **Présentation générale du problème** :

Ce projet se positionne dans le domaine de la reconnaissance des formes appliquée sur des images (scans, photographies...) de divers types de documents (partitions musicales, livres/manuscrits anciens, schéma électrique...). Il s'agit ici de concevoir une méthode de reconnaissance et de localisation d'éléments de contenu (EdC) (symboles, textes, formes...) applicable sur tout type de document.

Les deux objectifs principaux de ce projet sont, d'une part de déterminer une modélisation adaptée et efficace de la représentation de l'image d'un document (comment stocker les méta-données associés à un document ou à un modèle de document ? Que faut-il stocker ? Comment modéliser un document afin de pouvoir rapidement localiser un élément spécifique ? Comment afficher de manière claire et compréhensible ces informations ?). Et d'autre part de définir une méthode de description des éléments de contenu à reconnaître ainsi que leurs règles de localisation (Quels sont les paramètres discriminants ? Quelle structure adopter ?).

– **Domaine, marché et objectif** :

L'application sera la propriété de l'équipe RFAI ainsi que de leurs auteurs. L'objectif de ce logiciel est de pouvoir extraire aisément des règles caractéristiques dans des images, pour ensuite avoir la possibilité de localiser ou de segmenter des données. Les domaines d'application du logiciel sont assez nombreux, en particulier dans les domaines utilisant des images de documents, mais ce traitement pourra éventuellement être appliqué sur d'autres types d'images.

– **Maintenance** :

La maintenance du système sera principalement effectuée par Julien MICHOT, concepteur et développeur du logiciel.

1.3 Description de l'environnement

– **Entités** :

Les utilisateurs du système pourront être multiples : étudiants ayant accès à la section privée de l'équipe RFAI, les membres de l'équipe RFAI, d'autres utilisateurs externes ? Le logiciel, ainsi que les concepts proposés par celui-ci devront être très simples et intuitifs afin de pouvoir être manipuler par tous (experts et néophytes en informatique).

– **Informations** :

Le logiciel devra faire appel à un autre programme nommé **vecto_grapheJY.exe**

afin de calculer une vectorisation des images d'entrée. Ce logiciel requière une image *bmp*, en *noir et blanc* en entrée, et fournit en sortie un fichier **RvectoJY.dat**, dont le formalisme des données est fixe. Toute modification importante de ce formalisme se reportera sur la stabilité de notre logiciel.

– **Contraintes** :

L'application ne traitera que des images dont le type est connu (jpeg, bmp, tif, png...) et pourra fonctionner sur différents systèmes d'exploitation (Microsoft Windows, Linux, MacOS), seule la vectorisation ne pourra s'effectuer que sur les plateformes Windows.

1.4 Description des besoins à satisfaire

Nous allons voir dans cette partie la liste non exhaustive des fonctions principales à satisfaire dans le cadre de ce projet. Cette liste n'est bien entendue pas fixe, elle évoluera au cours du temps selon les besoins.

– **Fonctions principales** :
L'application devra être en mesure de :

1. lancer et récupérer les informations délivrées par le logiciel de vectorisation **vecto_grapheJY.exe**
2. créer la représentation de l'image à partir des primitives de description définies dans la modélisation choisie : affichage, stockage, navigation *(représentation intermédiaire : graphe)*
3. créer/modifier/enregistrer des successions d'actions (scénarios), en définissant des contraintes internes (liées aux caractéristiques de l'élément) et externes (liées au voisinage et au contexte).
4. appliquer un scénario sur un ou plusieurs documents afin de localiser les éléments de contenu
5. rechercher un EdC dans les méta données structurées (graphes valués) associés à une image ou a un ensemble d'images *(comparaison, recherche dans un graphe)*
6. construire automatiquement des éléments de contenu à reconnaître, à partir d'un ensemble d'images d'apprentissage.

7. fournir un fichier de sortie correctement défini et répertoriant les positions des différents EdC reconnus dans le(s) document(s), ainsi que leurs relations.

– **Caractéristiques et performances** :

Les caractéristiques des règles de description et des symboles doivent donner être générales, et aussi précises que possible. L'utilisateur doit pouvoir cibler le plus possible un objet en particulier, quelque soit sa forme ou sa position dans l'image. Le temps de calcul est dépendant de la taille de l'image et de sa complexité. Nous devons de plus compter le temps de calcul de la vectorisation, qui nous est imposé en tant que prétraitement.

1.5 Contraintes

Enfin, décrivons maintenant les contraintes du système.

– **Contraintes d'interfaces** :
Toutes les interfaces développées dans ce projet doivent répondre aux règles élémentaires d'IHM pour garantir un maximum de convivialité lors de leur utilisation. Une modélisation simplifiée pourra éventuellement être conçue si le choix de la représentation finale s'avère être trop complexe.

– **Contraintes de temps** :
Le temps imparti pour la réalisation du projet est d'environ 9 mois à compter de la date de remise des sujets. 4 mois supplémentaires seront ajouté lors du stage de master 2 de Recherche.

– **Contraintes d'utilisation** :
Le logiciel final devra être fonctionnel et utilisable sur tout ordinateur pourvu du système d'exploitation Microsoft Windows.

– **Contraintes de réalisation** :
Le logiciel sera développé en langage C++, à l'aide d'une interface Qt4. Celui-ci devra respecter les contraintes de réutilisabilité des classes et des fonctions.

– **Maintenance et évolutivité** :

Conformément aux contraintes de réalisation, l'évolutivité de l'application passe par la réutilisation des différents objets développés dans le cadre du projet. Les fichiers de sorties devront aussi respecter des normes standards (notamment XML) afin de pouvoir être réutilisés par un logiciel futur, par exemple.

– **Fiabilité et sûreté de fonctionnement** :
Le système devra bien entendu être le plus fiable possible. Les algorithmes développés devront fournir de bons résultats, pour la grande majorité des images de documents fournies en entrée. Le caractère généraliste des algorithmes et des méthodes de traitement est par conséquent un point important du projet.

1.6 Plan de développement

Le plan de développement a fait l'objet d'un document spécifique : SymbSpot_PDevPaq_003. Le planning de développement (figure 1.1) provient de ce document. Celui-ci est décomposé en deux parties PFE et M2R.

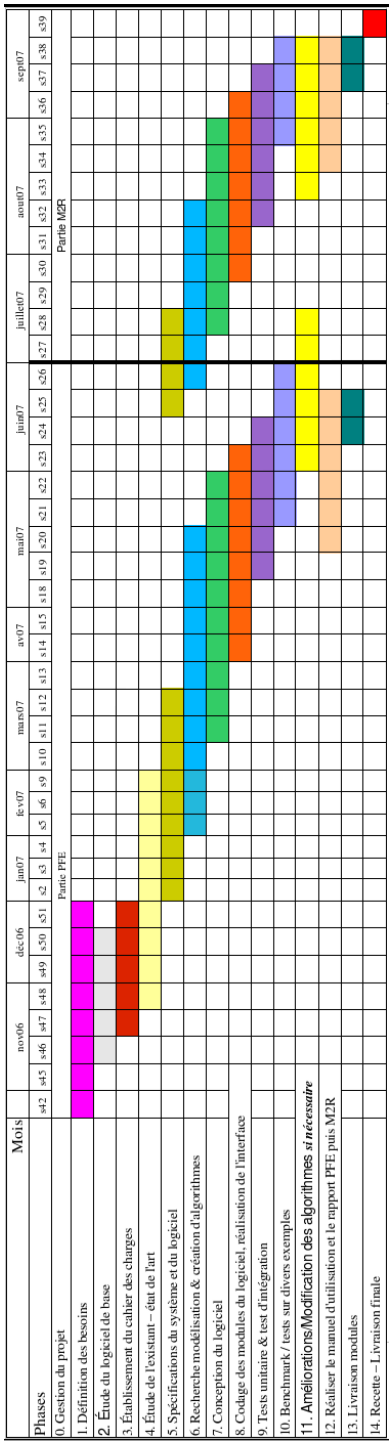


Fig. 1.1 – Planning prévisionnel de développement

Chapitre 2

Analyse et spécifications des besoins

Nous allons dans ce chapitre détailler les spécifications du système en analysant les besoins et les attentes des futurs utilisateurs. Nous présenterons de plus les algorithmes que nous avons implémenté dans le cadre du PFE et du stage de dernière année.

2.1 Introduction

Le Symbol Spotting est une technique d'indexation basée sur la recherche de formes connues dans un ou plusieurs documents. Ces formes sont déterminées par des règles de constitution plus ou moins restrictives, mais aussi par des règles de voisinage, de contexte, ou d'autre nature encore.

Deux principales méthodes seront testées au cours de ce projet. La première consiste en l'application de règles de description et d'action en vu de l'extraction d'éléments de contenu évolués. La seconde méthode sera quant-à-elle plus orientée sur une comparaison de graphes, plus précisément sur une recherche de sous-graphe (isomorphisme de sous-graphes).

2.2 Spécifications générales

Pour tenter de répondre aux différentes contraintes du projet mentionnées dans le chapitre précédant, il est nécessaire de se poser les questions suivantes :

- Interaction Homme-Machine : comment se fera le dialogue entre l'utilisateur et le logiciel ?
- Paramètres et caractéristiques des EdC : quelles seront les paramètres et propriétés des EdC accessibles au traitement ?
- Les entrées-sorties du système : quel est le formalisme des entités de stockage et/ou de dialogue ?

La suite de ce chapitre présente les éléments de réponse pour chaque problème posé et définit les principales entités constituant la base du projet.

2.2.1 Interaction Homme-Machine

Les scénarios

Les scenarii sont des fichiers regroupant les différentes actions de traitement, classées dans un ordre chronologique, que le logiciel devra suivre. Ces fichiers générés par l'utilisateur sont les fils conducteurs du traitement.

Il sera nécessaire donc de reprendre et d'intégrer la notion de scénario dans le logiciel. Il faudra de plus modifier le formalisme des fichiers enfin de les rendre plus exploitable, et notamment passer au format XML. Dès lors, nous pourrons aisément intégrer dans ces fichiers des définitions d'EdC (et non seulement des règles de traitement/action), et ainsi constituer par exemple des dictionnaires.

Le logiciel devra aussi permettre l'import et l'export des scenarii, ainsi que l'édition, le traitement par lot et l'appel à l'exécution d'autres scenarii.

L'ensemble nous donnera la possibilité de constituer des dictionnaires propres à un type de document par exemple, et de pouvoir l'intégrer dans un scénario de traitement pur.

Création de règles

Les règles de traitement sont la base de la première méthode d'extraction d'éléments. Celles-ci donneront la possibilité aux utilisateurs de nommer de nouveaux éléments de contenu, de supprimer ou de fusionner des ensembles hétérogènes.

De plus, la création des règles (de description des EdC et d'actions) se fera dans un 1er temps à l'aide de boites de dialogues. Celles-ci devrons être simples et compréhensibles par des néophytes, mais devrons aussi permettre des traitements plus complexes.

Un module d'apprentissage sera ensuite implémenté afin de diminuer l'implication de l'utilisateur lors de la création des EdC. Ce module permettra alors à tout néophyte de décrire des EdC en ne présentant que des images des EdC et non en ajoutant des règles plus ou moins explicites. Il déterminera donc la meilleur représentation sous forme de graphe de l'EdC appris.

Représentation finale

Une fois l'ensemble des traitements opéré, le logiciel fournira un fichier de sortie reprenant l'ensemble des EdC reconnus, ainsi que des informations de localisation (voir la partie Entrées-Sorties de ce chapitre). Ce fichier, certes très informatif, reste un fichier XML peu visuel, nous avons donc envisagé d'ajouter au projet un module de visualisation qui, à partir des informations de sortie du logiciel, dessinera les zones reconnues (rectangle englobant avec étiquetage notamment).

Il pourrait aussi être intéressant de pouvoir sélectionner les symboles que l'on désire afficher, afin de ne pas surcharger la visualisation.

2.2.2 Éléments de contenu et règles

Étant donné que tout document peut être défini comme en ensemble d'éléments de contenu sur un support [2], nous nous intéressons donc à leur reconnaissance et leur localisation de le document. Cette étape pourra nous permettre ensuite par exemple d'indexer des documents par leur structure physique et leur contenu, plus ou moins évolués suivant le taux de confiance de la reconnaissance. La description des éléments de contenu se réalisera premièrement au travers de règles portant sur divers paramètres géométriques, géographique et de contexte (voisinage). Nous allons ici nous intéresser aux différentes propriétés accessibles à l'utilisateur et au logiciel.

Liste des caractéristiques physiques des EdC

- Position du centre de gravité de l'EdC dans le document (**X** : entier ,**Y** : entier)
- Bounding Box : (longueur **L** : entier , largeur **l** : entier)
- Minimum Bounding Rectangle : (position des points)
- Angle **a** : définissant l'axe d'inertie de l'EdC
- Paramètres statistiques (densité des niveaux de gris, couleurs, nombre de primitives élémentaires,...)
- Le type d'EdC

L'accès à ces données sera effectué à l'aide de champs liste permettant de définir des relations binaires (entre deux paramètres) (ex : $\frac{L}{l} < 2$) ;

Liste des relations possibles (interne et de voisinage extérieur)

Après avoir défini les caractéristiques générales de l'EdC, il faut bien entendu pouvoir préciser les relations internes de voisinage des EdC. La liste ci-dessous correspond aux propriétés relationnelles inter-EdCs (entre un EdC quelconque X et un EdC quelconque Y) pour la définition de la composition des EdC plus évolués.

- relations directes : définies par un ensemble de propriétés relationnelles ou physique
  - X est à gauche de Y (pas forcément le 1er arc)
  - X est à droite de Y
  - X est au dessus de Y
  - X est en dessous de Y
  - (X, Y) sont à une distance inférieure/supérieure/à peu près égale à ?
- relations de superposition :
  - X est totalement ou partiellement inclus dans Y (les Bounding Box se recouvrent), avec par exemple une fonction Intersection(X, Y) retournant le pourcentage de recouvrement des Bounding Box des deux EdC.
- relations d'angle
  - (X, Y) forment un angle inférieur/supérieur/à peu près égale à ?
  - X est perpendiculaire à Y (T)
  - X est parallèle à Y (P)
  - X forme un L avec Y (L)

Un EdC pourra alors être décrit par un graphe interne de voisinage d'EdC qui le compose récursivement, en précisant sur les noeuds le type d'EdC ainsi que leurs caractéristiques physiques propres et sur les arcs, les relations de voisinage inter EdC.

La modélisation du graphe ainsi que le choix des prédicats et des fonctions relationnelles seront précisés lors dans le chapitre de conception générale et détaillée.

Construction d'une règle d'extraction d'un EdC

Un fois que l'ensemble des contraintes est parfaitement défini par l'utilisateur (en utilisant les caractéristiques mentionnées précédemment), le choix d'une action est alors proposé.

- La Fusion : Création d'un nouveau EdC (en précisant les EdC à fusionner)

– Changement de type : Tous les sous EdC seront alors vus comme une instance du type sélectionné.
– Suppression : éliminer du graphe et de la mémoire les sous EdC sélectionnés

Lorsque l'utilisateur désire définir un nouvel EdC, quelques paramètres d'enregistrement lui sont alors proposés : l'enregistrement des instances trouvées par l'algorithme dans le fichier de sortie, et le choix de la couleur d'affichage.

Pour illustrer nos propos, prenons l'exemple d'un utilisateur désirant extraire un type d'EdC nommé Condensateur. Pour le modéliser, l'utilisateur devra préalablement imaginer et nommer les constituants en vu de leur extraction. Il s'agit ici d'une étape très importante. La définition des EdC doit permettre leur reconnaissance dans les documents et doit donc être assez discriminante pour classifier l'élément et le séparer des autres (bruit). Mais les règles de définition doivent aussi prendre en compte le reste du document ! Par exemple, définir un EdC nommé Condensateur par la seule règle 'deux quadrilatères parallèles' permet certes de trouver les Condensateurs réels, mais elle détecte aussi des générateurs (ou plus généralement, les symboles proches) et les considère donc comme des Condensateur. Un autre ensemble de règles semble être plus discriminant :

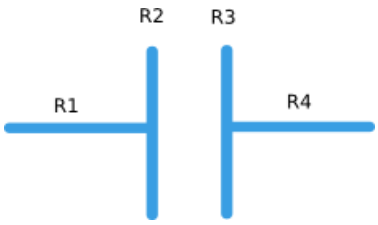


FIG. 2.1 – Modélisation d'un condensateur

L'utilisateur pourra par exemple définir l'élément par les règles suivantes :
– Liste des propriétés physiques : *par exemple : nombre de primitives élémentaires* < 4*, etc...*
– Liste des EdC : *R1 :Quadrilatère, R2 :Quadrilatère, R3 :Quadrilatère, R4 :Quadrilatère*
– Liste des relations :
  – *R1* est perpendiculaire à *R2* (T)
  – *R1* est à gauche de *R2*
  – *R3* est perpendiculaire à *R4* (T)
  – *R4* est à droite de *R3*
  – *R2* est parallèle à *R3* (P)

- *R2* et *R3* ont la même longueur.
- Action : Créer un EdC en fusionnant *R2* et *R3* , si l'on désire garder les fils de liaison de gauche et de droite ou dans le cas échéant : Créer un EdC en fusionnant le tout.

Cette règle de description du condensateur n'est pas unique. Nous pouvons définir une multitude de règles permettant de décrire un condensateur. Par exemple, une règle définissant les éléments parallèles en un nouvel EdC (*edp2*), puis une autre règle spécifiant que 2 *quadrilatères* ayant une liaison en L avec un même élément de type *edp2* deviennent un élément de type *Condensateur*...

La modélisation graphique du condensateur peut alors prendre la forme suivante :

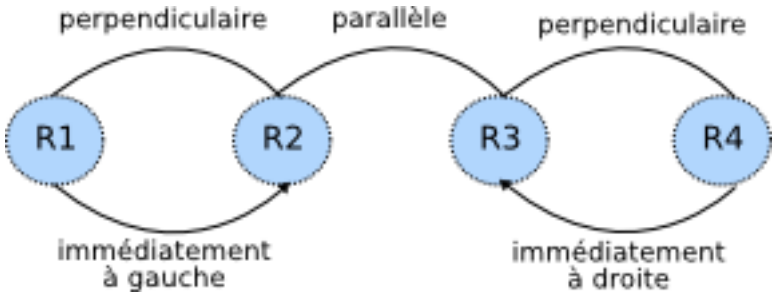


FIG. 2.2 – Graphe de modélisation du condensateur

2.2.3 Spécifications entrées-sorties

Nous allons nous intéresser ici aux spécifications des flux d'entrée et de sortie du logiciel.

Fichiers d'entrée

Au niveau des formats des fichiers d'entrée, nous employons les fonctionnalités du logiciel de base pour l'ouverture des images (formats d'images standards : jpeg, bmp, png, tiff). En ce qui concerne les règles d'action et de description des EdC, nous devons formaliser leur gestion sous le format XML.

Fichiers de sortie :

Les fichiers de sortie sont de deux types : les scenarii (entrée-sortie) et les fichiers résultants du traitement. La spécificités des scenarii sont définies dans la partie 2.1.1.1

de ce document. Le fichier présentant les localisations des EdC devra quant à lui répondre aux contraintes suivantes :

- Lister les EdC reconnus avec :
  - Le type d'EdC (ID, URI)
  - Le nom du document image traité (NOM du fichier)
  - La position géographique dans ce document (X :entier,Y :entier)
  - La Bounding box
  - Le taux de confiance
- Préciser des informations supplémentaires :
  - Le nom du fichier contenant la liste des règles d'action et de description d'EdC appliqué (pour avoir la définition des EdC)
  - Le temps du traitement du document
  - Éléments statistiques (nombre d'EdC reconnus...)

Il pourra aussi être souhaitable de pouvoir ouvrir les fichiers résultats, fournis par notre algorithme afin de modéliser graphiquement le résultat de la recherche.

Nous utiliserons le formalisme GXL pour tous les fichiers modélisant des graphes, soit pour le fichier de localisation final ainsi que pour les fichiers de graphe modèle (cf. la partie 6 de ce document pour le détail du format sGXL).

Afin de mieux répondre aux attentes du projet, nous avons dressé une liste non exhaustive des algorithmes nécessaires à l'élaboration de ce logiciel.

2.3 Liste des algorithmes nécessaires

Algorithme de construction du graphe initial de l'image

Le premier algorithme qui sera étudié est l'algorithme de construction du graphe initial. A partir d'un ensemble composé de quadrilatères, de vecteurs, de composantes connexes, ainsi que de relations inter-quadrilatères, l'algorithme devra concevoir le graphe initial du document. Celui-ci fournira dès lors un graphe initial que l'on manipulera tout au long des traitements. Ce graphe devra être parfaitement générique du point de vu des noeuds, mais aussi être optimisé pour la recherche et le parcours des noeuds/arcs.

Algorithme de gestion de scenarii et des modèles

Nous allons nous intéresser ici à l'implémentation d'une gestion de fichiers xml afin d'intégrer le stockage des EdC, ainsi que des règles d'action définies par l'utilisateur. Cette gestion sous forme de fichiers exportables donnera la possibilité aux utilisateurs de constituer par exemple des dictionnaires représentant le langage du document (syntaxique et sémantique). L'application devra donc gérer toutes les fonctions de base de gestion d'éléments comme notamment : l'import, l'export des scenarii, l'appel interne à un autre scénario, l'ajout, la suppression d'éléments...

Premier algorithme de symbol spotting

Après avoir conçu le graphe modélisant le document à traiter, les règles de définition des EdC que l'on souhaite rechercher et les règles d'action, nous pouvons commencer la recherche de symboles.

Nous proposerons une méthode adaptée aux propriétés et aux contraintes de la représentation de l'image, pour la recherche de symboles définis par des règles. L'algorithme principal sera fondé sur l'application de règles de décision (géométriques, géographiques et de voisinage) afin de classifier dynamiquement les EdC. Le résultat de la première itération sera ensuite retravaillé en vue de déterminer/localiser des EdC plus évolués et ainsi de suite. Le logiciel s'arrêtera lorsque plus aucune règle ne peut être appliquée, ou lorsque l'utilisateur en fait la demande.

L'algorithme précis est défini dans le chapitre de conception du projet.

Deuxième algorithme de symbol spotting

La deuxième méthode de reconnaissance et de localisation d'EdC sera quant à elle basée sur un appariement de graphes []. Pour améliorer la rapidité du traitement (et notamment avec les relations de voisinage), l'algorithme utilisera les propriétés fondamentales des graphes (optimalité de la recherche de sous graphes : subgraph matching). L'idée d'une recherche de sous graphes dans un graphe avec tolérance pourra éventuellement être exploitée.

Algorithme de calcul et propagation de caractéristiques dans un graphe

Lorsqu'une règle est validée, une action est alors entreprise par le logiciel. Lorsqu'il s'agit d'une fusion (création d'un nouveau EdC) certaines propriétés du graphe de

l'image doivent être recalculées, d'autres non. On peut notamment effectuer les étapes suivantes :

- Recalculer des paramètres physiques (bounding box englobante, centre de gravité, angle d'inertie...) [noeuds]
- Vérifier et garder des relations spatiales (gauche/droite/...) [arcs]
- Autres...

Cet algorithme est présenté dans ce document dans le chapitre de conception générale.

Algorithme d'affichage du symbol spotting

En complément du fichier xml de sortie localisant les différents EdC reconnus, un module d'affichage du résultat sera ajouté au système afin de voir graphiquement ce qui est reconnu et localisé. Il s'agit ici d'une simple lecture du fichier de sortie pour ensuite réaliser l'affichage coloré des EdC.

Parser XML

Puisque la grande majorité des fichiers d'entrée et de sortie seront sous la forme XML, nous devons intégrer dans ce projet un module de lecture-écriture de fichiers XML.

Chapitre 3

Définitions et conception des éléments de la représentation d'images sous forme de graphe

La modélisation des documents est une étape essentielle du projet. Elle est en effet la source sur laquelle toutes les détections et traitements futurs seront effectués. Il s'agit donc de définir un modèle structuré que l'on pourra manipuler, traiter, modifier et comparer le plus simplement possible. Le modèle devra fournir de nombreux paramètres de position, de géométrie et de voisinage et devra être en mesure de répondre aux évolutions futurs du projet.

3.1 Modélisation sous forme de graphe orienté multivalué

L'ensemble des contraintes précisées dans le cahier des spécifications nous a amené vers une solution basée sur les graphes, comme dans [4] . Les informations de caractérisation des documents seront donc structurées en graphes multivalués (sur les noeuds et sur les arcs) et multi-niveaux (2 ou plus). Nous modéliserons les éléments de contenu par les noeuds (vertex) du graphe, les arcs (edges) seront quant-à-eux utilisés pour stocker les relations de voisinage inter-EdCs. Les niveaux du graphe représentant les niveaux des éléments reconnus (un quadrilatère étant de niveau inférieur à un Condensateur par exemple). Pour un état de l'art plus complet, se référer au très bon document de Delalandre [6].

Lors de la première étape du logiciel, c'est-à-dire une fois l'étape de traitement et de reconnaissance de bas niveau effectuée (vectorisation), les EdC présents dans le graphe seront de trois types distincts : les Composantes Connexes, les Quadrilatères et les Vecteurs. L'ensemble des arcs sera de plus constitué par les relations inter-quadrilatères fournis par le programme VectographJY. La modélisation primaire nécessite une quantité limitée de paramètres (noeuds et arcs), mais puisque la modélisation (les EdC) du document évoluera au cours du traitement, nous devons définir de nouveaux paramètres supplémentaires. Étant donné que les vertex du graphe représenteront des EdC, les paramètres devront être parfaitement génériques et aisément calculables quelque soit le type ou la complexité de l'EdC représenté.

L'objet abstrait Noeuds (ou Vertex) contiendra les paramètres suivants :

- Type d'EdC **type** : l'ID du type d'élément de contenu
- Centre **c :(X,Y)** : le centre de gravité de l'EdC, défini comme le milieu de la Bounding Box
- Bounding Box **BB :(xmin,xmax,ymin,ymax)** : la bounding box englobant l'élément
- Bounding Rectangle **BR :(P1,P2,P3,P4)** : le rectangle englobant de l'élément
- Angle principal **ai)** : défini par l'angle du rectangle englobant, et représentant l'angle d'inertie de l'EdC
- Densité des niveaux de gris **densité** : densité des NdG de la BB
- Couleur **c** : couleur moyenne de l'EdC
- Liste des sous EdC primaires que contient l'élément
- Nombre de primitives élémentaires **n** : le nombre de sous-EdC constituant l'élément
- Taux de confiance **tc** : le taux de confiance de la reconnaissance de l'élément (à définir, pour l'algorithme d'apprentissage)

Le schéma suivant illustre des différentes éléments et propriétés ajoutés à l'EdC.

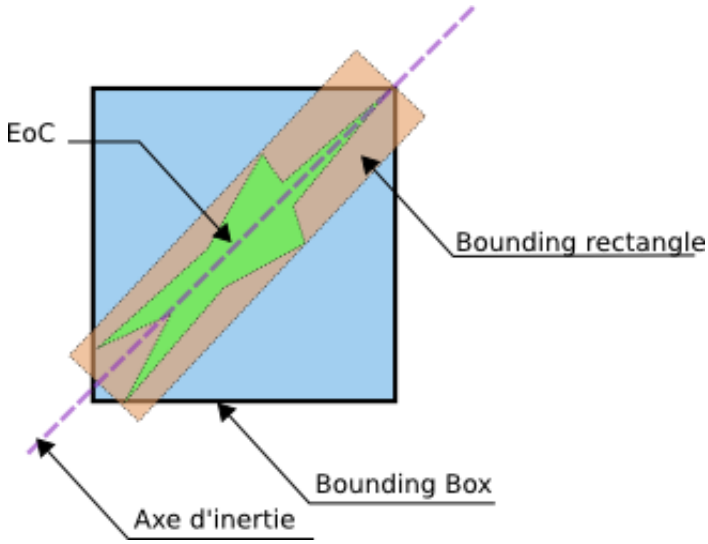


FIG. 3.1 – Approximations d'un élément de contenu EoC

Les Arcs (ou Edge) orientés modéliseront les relations de voisinage entre deux EdC, et posséderont les propriétés suivantes :

- Distance minimale **dmin** : distance minimale entre les deux EdC (entre les Rectangles Englobants)
- Angle inter EdC **a** : angle entre les deux axes d'inertie des deux EdC
- *orientation* : ∈ { Gauche,Droite,Dessus,Dessous,Inclus,Partiellement inclus,L , l , T , t , P , X , s ,Indéfini }

Ces dernières relations permettrons de ne pas recalculer les propriétés nécessaires à chaque règle de traitement. Il est en effet plus intéressant de calculer ces valeurs à chaque détection d'un nouvel EdC que de le faire lors de la recherche. De plus, les relations d'angle du type L,T, etc... ne seront visibles que dans la modélisation primaire du graphe, et éventuellement, à plus haut niveau.

3.2 Approximation des EdCs et détermination de l'axe d'inertie

Dans l'optique d'utiliser le logiciel sur divers types de documents, une abstraction des éléments du document est nécessaire. De plus, afin de pouvoir déterminer des relations inter-EdC, nous devons utiliser une forme approximée de l'élément.

Il existe plusieurs types d'approximation dans la littérature. Dans le domaine de la réalité virtuelle (3D) par exemple, on utilise beaucoup les bounding box (boîtes englobantes), mais on peut tout aussi bien prendre une approximation convexe de la forme. Dans notre cas, nous avons choisi d'implémenter trois types d'approximation : la boîte englobante, le rectangle englobant, ainsi que l'enveloppe convexe. Le rectangle orienté englobant (Minimum enclosing rectangle) sera principalement manipulé, notamment pour la détection d'inclusion ou pour le calcul de la densité de pixels noirs.

Pour calculer le rectangle englobant d'aire minimal, nous avons fait appel à l'algorithme Rotating Calipers.

Rotating Calipers

La méthode des rotating calipers est une méthode assez ingénieuse pour calculer le rectangle englobant d'aire minimal. L'algorithme de Toussaint [1] consiste à faire tourner deux droites autour de l'élément convexe comme le montre la figure suivante.

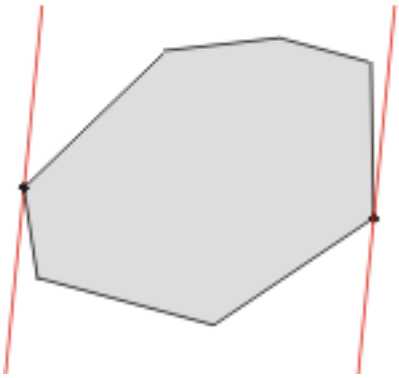


FIG. 3.2 – Rotating Calipers

La complexité de la méthode (O(n)) est ainsi réduite, puisqu'elle ne dépend principalement que du nombre de points de l'élément convexe, ce qui dans notre cas n'est pas très important.

Une fois le rectangle englobant déterminé, nous pouvons calculer l'axe principal du rectangle (en prenant par exemple l'un des segments du rectangle (dans le sens de la longueur)), et nous pouvons dès lors avoir l'angle principal de l'élément de contenu.

Voici le diagramme de classe représentant la méthode et les objets employés pour le calcul des Rotating Calipers.

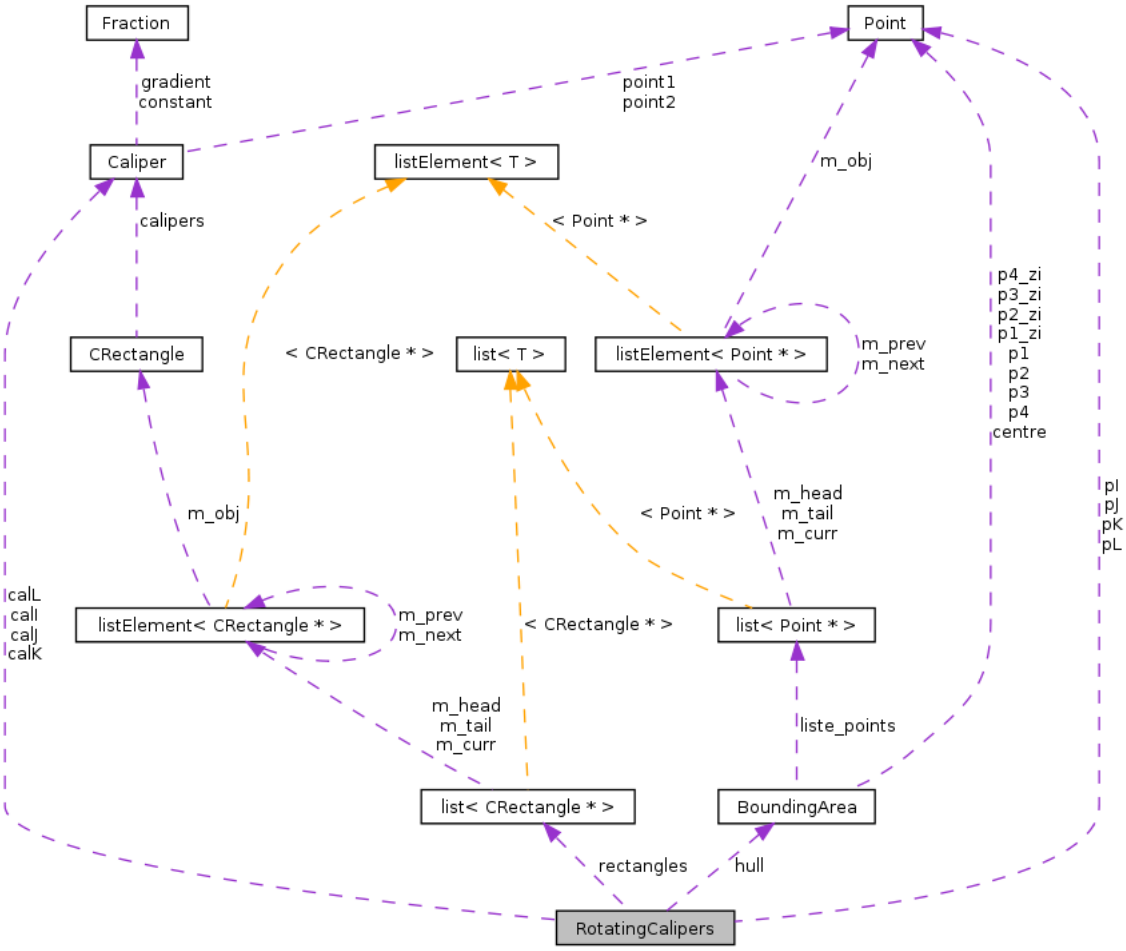


FIG. 3.3 – Rotating Calipers

Après avoir cherché à implémenter la méthode, nous avons choisi d'employer la librairie de calculs géométriques CGAL. www.cgal.org. Cette bibliothèque c++ implémente de nombreuses méthodes très utiles et pour la majorité d'entre-elles le code ainsi que les algorithmes employés sont optimaux (ou dans le cas échéant, optimisés).

Le calcul du rectangle englobant fournit une valeur d'angle qui représente l'angle général de l'élément. Cette valeur n'est cependant pas l'angle d'inertie réel de l'objet, mais il s'agit plutôt d'un angle approximé.

ACP

Pour calculer l'angle d'inertie d'un objet en 2 dimensions, il nous faut faire appel à une ACP[1]. Celle-ci consite à déterminer l'axe principal d'un ensemble de points, ce

[1] Analyse en Composantes Principales

qui nous permet donc ensuite d'avoir l'angle d'inertie.

Or, le calcul de l'ACP est une étape assez longue, d'autant que nos ensembles de points sont en fait, l'ensembles des points appartenant à l'élément de contenu. Par exemple pour un quadrilatère plein, on obtient un nombre important de points. Un échantillonnage adapté pourrait semble-t-il réduire les calculs.

L'analyse en composantes principales est déjà implémentée au sein de la librairie CGAL. Pour cela, nous utilisons la fonction **linear_least_squares_fitting_2** qui calcule la droite minimisant la sommes des distances.

3.3 Distance minimale entre deux EdCs

Nous nous intéressons dans le cadre de la définition du voisinage des EdCs au concept de distance inter-EdCs. Dans le but d'être le plus générique possible, nous proposons d'utiliser la distance minimale entre deux EdCs. Cette distance répond à de nombreux problèmes auxquels une simple distance entre centroïdes n'aurait pu être en mesure de résoudre.

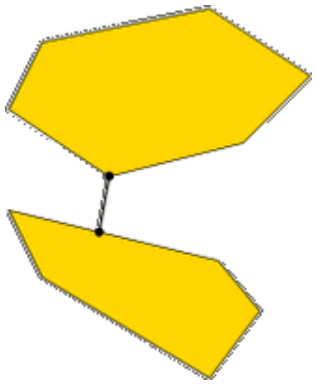


FIG. 3.4 – Distance entre deux polygones

Le problème de la recherche de la distance entre deux polygones convexes peut se formuler comme un problème d'optimisation pourvu de contraintes linéaires et d'une fonction objective quadratique. La solution du problème est obtenu à l'aide d'un solveur quadratique [3], intégré à la librairie CGAL.
Nous employons cet algorithme sur l'ensemble des points extrêmes des éléments de contenu. Il se chargera alors de calculer la distance minimale des formes convexes des deux EdCs. La fonction appelée est **Polytope_distance_d** .

3.4 Calcul de l'angle inter-EdCs

La deuxième valuation présente sur les arcs relationnels est l'angle formé entre les deux éléments. Cette information nous semble être assez intéressante, notamment pour la détection de symboles graphiques électroniques.
Puisque nous connaissons les axes d'inertie des deux éléments, il semble être assez simple de calculer cette valeur, or, il faut remarquer que deux cas sont possibles. La figure suivante montre que suivant la position des éléments, l'angle qui nous intéresse diffère.

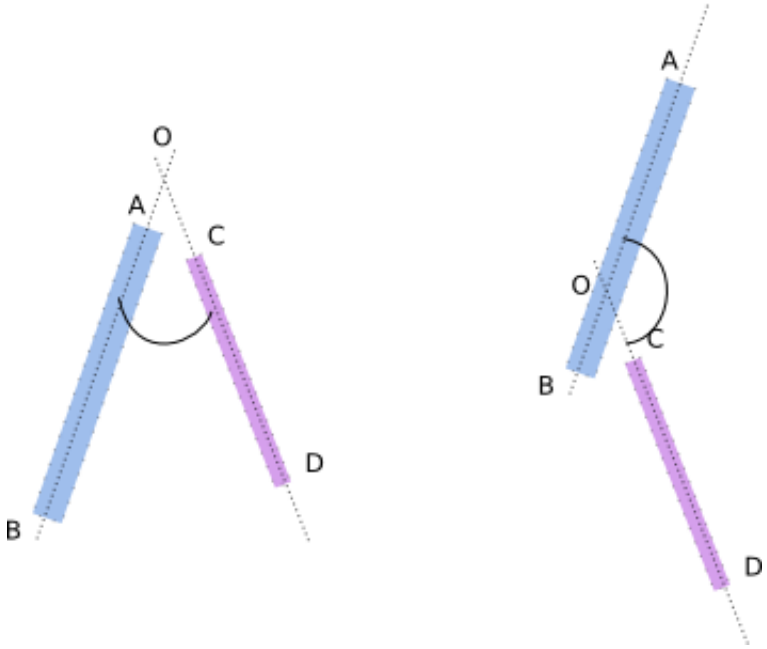


FIG. 3.5 – Angles entre deux EdCs, 2 cas possibles

Pour différencier les deux possibilités, nous nous sommes basé sur la position du point d'intersection O des deux axes d'inertie. Si ce point est plus proche du point A, alors nous sommes dans le premier cas, sinon, nous sommes dans le deuxième.

Par défaut, le logiciel calcule les angles des EdCs de telle manière à obtenir des angles entre -90 et 90 degrés. Par conséquent, le calcul de l'angle inter-EdCs est simple :

cas 1 : produit scalaire entre [AB] et [CD]

cas 2 : produit scalaire entre [BA] et [CD]

3.5 Algorithme de construction du graphe de l'image

Pour construire le graphe modélisant le document, nous proposons l'algorithme suivant :

Algorithme 1 Algorithme de construction du graphe de l'image

Entrées: une image
Sorties: un graphe multivalué

1. Le traitement primaire (appel au programme de vectorisation vectographJY), nous permet d'obtenir un ensemble d'informations contenant :
→ une liste de vecteurs
→ une liste de composantes connexes
→ une liste de quadrilatères
→ une liste de relations entre deux quadrilatères
2. Création du graphe primaire (Noeuds/Arcs)
3. Calcul des Bounding Rectangles (méthode des Rotating Calipers) :
→ calcul du polygone convexe
→ tri des vertex dans le sens horaire
→ détermination des Calipers
→ retourner le rectangle d'aire minimale
4. Calcul des paramètres statistiques fixes (densités de pixels noirs, couleur...) et calcul des BB, Couleur,...
5. Détermination des Composantes Connexes auxquels appartiennent les vecteurs et les quadrilatères (comparaison de position)
6. Re-détermination des relations de voisinage entre tout les EdC (en utilisant les Bounding Rectangle de chaque EdC). Cette étape ne s'effectue qu'une seule fois lors de la création du graphe initial.

Afin de définir les relations de voisinage, il nous faut tout d'abord caractériser ce qu'est le voisinage d'un EdC. Il existe en effet une multitude de voisinage en traitement de l'image, et certains sont plus appropriés que d'autres suivant le type de problème considéré. Dans notre cas, nous manipulons des formes très diversifiées puisque les éléments de contenu sont parfaitement générique. Il convient alors d'utiliser une approximation, et notamment les bounding box, plutôt que la forme de base.

Les relations du type "est en dessous" ou "est à gauche de", peuvent être vérifiées par l'intermédiaire de différentes méthodes - nous en proposons 4 -, illustrées dans la figure suivante.

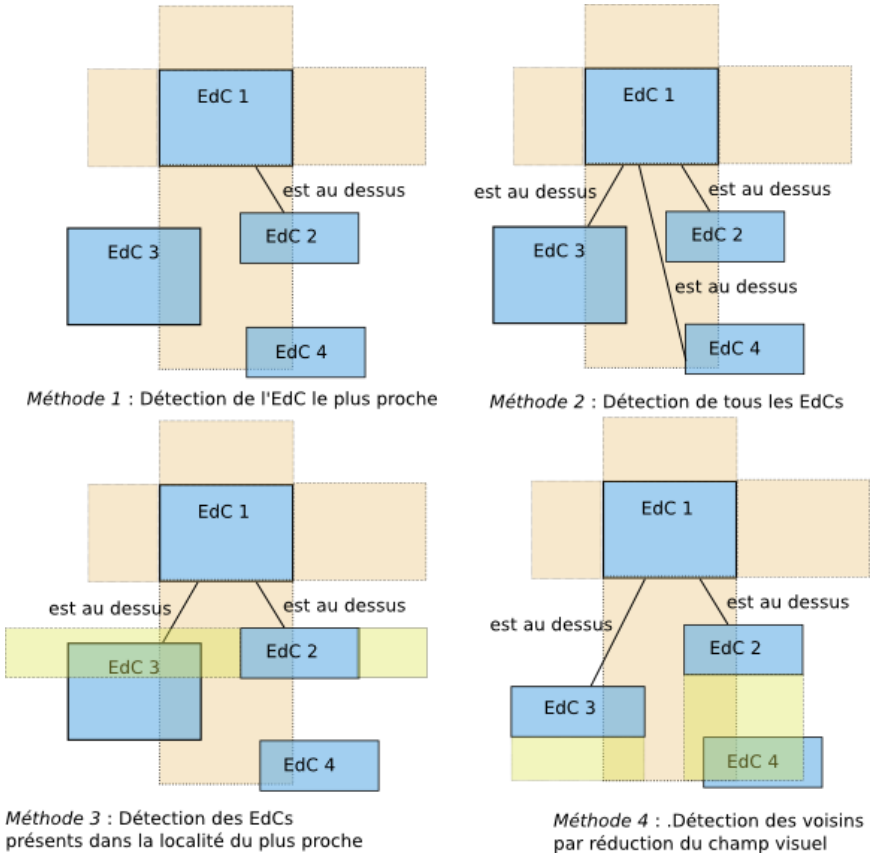


FIG. 3.6 – Propositions de détection de voisinage

La première méthode consiste à déterminer l'élément de contenu le plus proche de l'EdC courant, pour chaque direction (haut, bas, gauche et droite). Par exemple dans le cas précédent, un lien est défini entre EdC1 et EdC2. Cependant, l'élément EdC3 pourrait tout aussi bien être considéré comme "en dessous" de EdC1.

La deuxième proposition consiste à déterminer l'ensemble des EdCs présents dans les zones (haut, bas, gauche et droite) (avec une limitation de la distance). Nous obtenons alors beaucoup de relations, mais aussi et surtout des relations indirectes : par exemple, l'EdC 4 est considéré comme en dessous de l'EdC 1 alors que finalement il serait plutôt en dessous de l'EdC 2. Ces cas sont de plus très dépendants du seuil de distance fixé.

Une autre méthodologie consisterait à, d'une part déterminer l'EdC le plus proche pour chaque direction, et d'autre part, de ne prendre considérer que les éléments de contenu présents des les zones opposées de celui-ci. Dans l'exemple précédent, l'EdC 3 appartient à la zone de gauche de l'EdC 2 (qui est le plus proche de l'EdC 1). Cette méthode est relativement efficace, surtout si l'élément le plus proche est mince.

Enfin la proposition 4 consiste à définir des zones d'occlusion permettant d'éliminer

les EdCs non “visibles” pour chaque direction. Cette méthode semble être plus précise mais d'une complexité un peu plus importante puisqu'il est nécessaire de stocker la liste de domaines visibles ou non-visibles. Nous pourrions de plus considérer non plus des rectangles mais des parallélogrammes (triangles écrêtés) dont l'angle de restriction serait proportionnel à la distance avec l'EdC courant (principe de champ visuel).

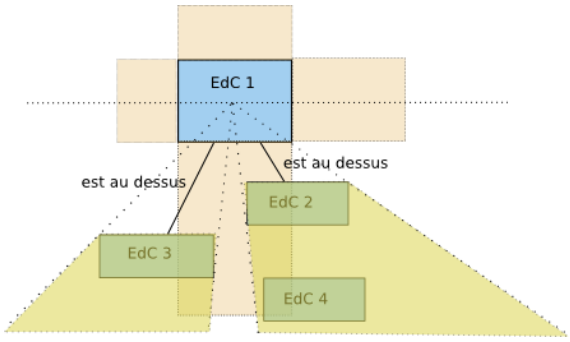


FIG. 3.7 – Occlusion par polygones - champ visuel

Ici, on utilise le champ visuel réel de l'EdC. Dans la figure suivant nous prenons une approximation de celui-ci afin d'éliminer les éléments trop éloignés.

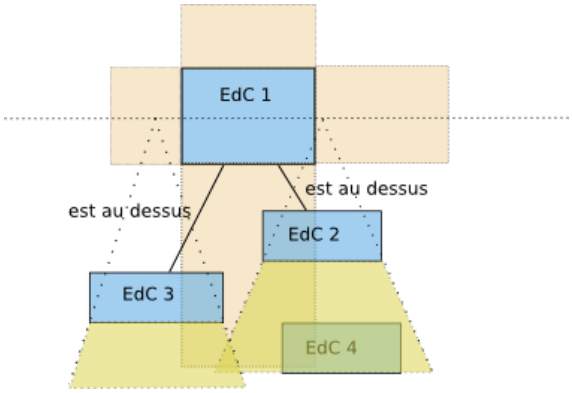


FIG. 3.8 – Occlusion par polygones - approximation

L'algorithme consisterait alors à vérifier que l'EdC n'intersecte pas les zones d'occlusion (pas de liste de domaines).
Les autres relations de voisinage serait définis de la manière suivante :

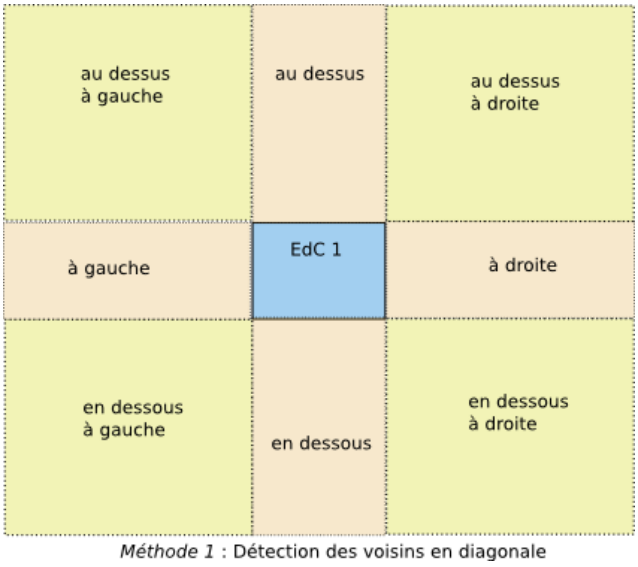


FIG. 3.9 – Détection des voisins - 8 connexité

La détermination de la présence d'éléments dans ces zones peut être effectuée par les algorithmes précédents.
La méthode de détection du voisinage à ce jour implémentée valide la proposition 4 de la figure 3.6.

3.6 Calculs de nouveaux paramètres, mise à jour des informations

Une fois la première modélisation de l'image effectuée, les EdC primaires seront amenés à évoluer au cours de l'application de scénarios. Les modifications et mises-à-jour des informations dépendent du type de règles employées.

Pour les règles de renommage, il s'agit simplement de changer le type de l'élément de contenu. Les autres paramètres étant parfaitement génériques, il n'est pas nécessaire de faire d'autres calculs inutiles.

Pour les règles manipulant plusieurs EdCs (comme la fusion), nous devons recalculer l'ensemble des paramètres du nouvel objet ainsi créé et mettre à jour les relations (arcs). La fusion d'un sous graphe en un nouveau noeud soulève la question suivante : que faire des arcs entrant et sortant du sous graphe ? Trois cas de figures sont possibles :

1. Mise à jour : un paramètre de l'arc a changé. Ce cas est le plus fréquent. Puisque

le centroïde aura changé de position, les distances sont à recalculer. Mais il faut aussi vérifier que les relations spatiales sont inchangées (Gauche/Droite/...)

2. Création : avec la fusion d'EdCs nous devons aussi redéfinir la zone d'influence (voisinage) de l'EdC. Il est dès lors possible que de nouveaux arcs apparaissent.
3. Suppression : cas envisageable mais non encore employé.

Chapitre 4

Modélisation UML des éléments du graphe

Dans la suite de cette partie nous présentons les différentes classes du projet. Pour être concis, nous avons choisi de mentionner les classes les plus importantes.

4.1 Graph

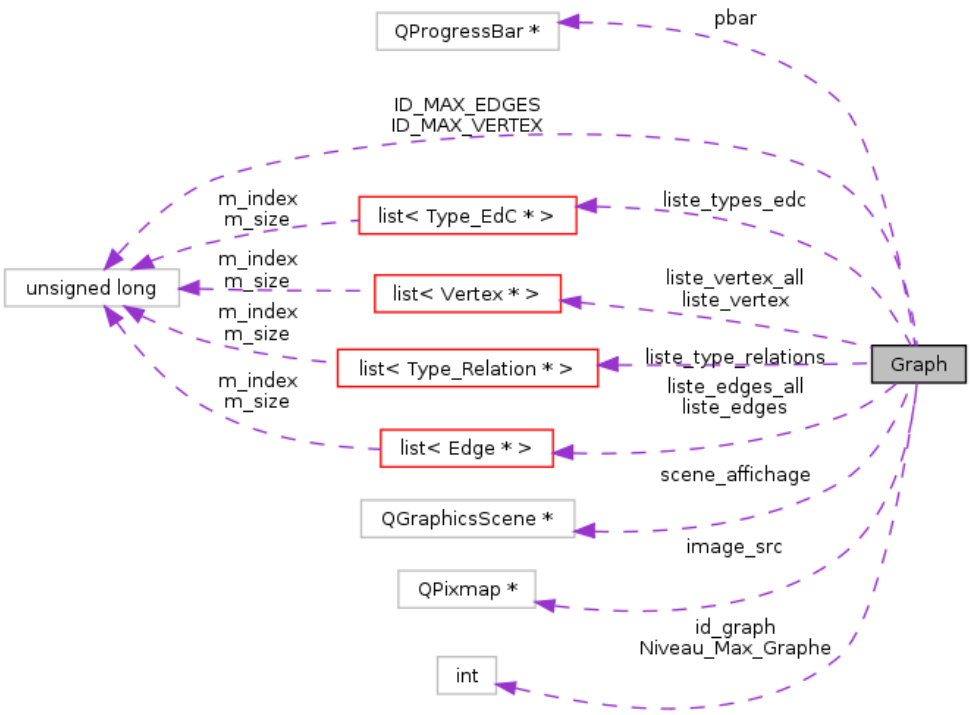


FIG. 4.1 – Diagramme de Classe - Graph

La classe Graph est l'une des classes essentielle du projet. Elle contient en effet toute la modélisation du document avec entre autres, la liste des EdCs (noeuds), la liste des relations inter-EdC (arcs), la liste des types d'élément de contenu connus du graphe ainsi que la liste des types de relations. Cette classe contient de plus un pointeur sur l'image source (non modifiée) et un pointeur sur un graphe de scène (affichage de sortie).

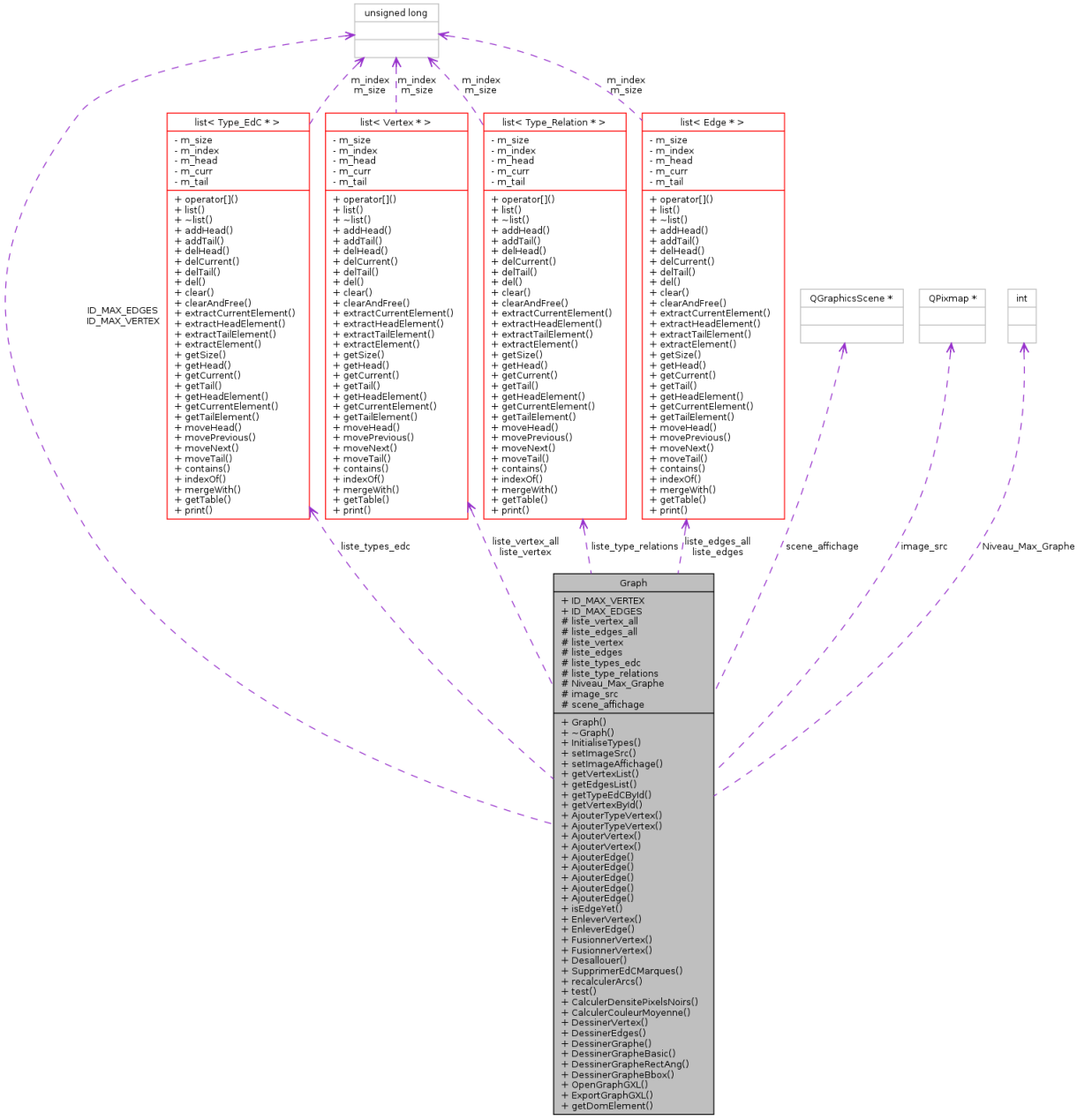


FIG. 4.2 – Modélisation UML- classe Graph

La classe Graph donne aussi accès à quelques méthodes d'affichage permettant

de dessiner les EdCs, les arcs et autres données utiles du graphe, et ce, de manière totalement indépendante les unes des autres. Elle possède de plus un indicateur de niveau du graphe correspondant au niveau du graphe courant. Celui-ci est employé lors de la fusion d'EdCs par exemple, puisque lors d'une fusion, nous conservons les sous-EdCs fusionnés ainsi que les arcs internes (entre 2 sous-EdCs). Cela nous permet donc de modéliser soit un graphe multi-niveaux, soit un graphe normal.
La classe Graph contient de plus des méthodes d'entrée-sortie GXL, qui permettent soit d'exporter directement l'ensemble du graphe en un fichier GXL, soit d'importer un graphe depuis un fichier GXL.

4.2 Vertex (ou EdC)

La classe vertex modélise un élément de contenu, ainsi que toutes les caractéristiques de celui-ci.

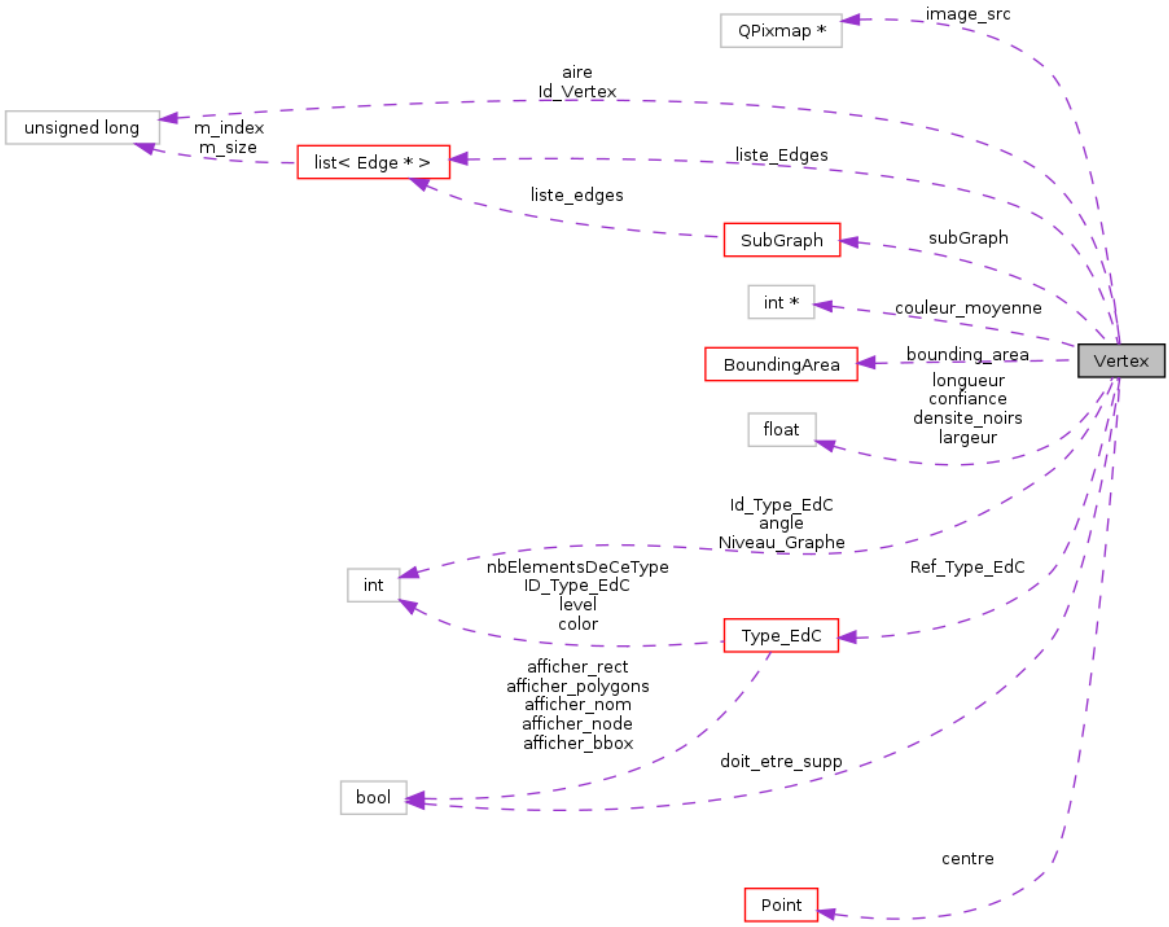


FIG. 4.3 – Diagramme de classe - Vertex

Cette classe contient un pointeur vers ses approximations géométriques (Bounding Area), un pointeur vers un sous-graphe éventuel, une référence sur son type d'EdC ainsi qu'une liste d'arcs entrants et sortants du noeud. Cette dernière liste nous permet un accès très rapide à tous les arcs du vertice et réduit ainsi la complexité de l'algorithme de fusion/recherche d'EdCs.

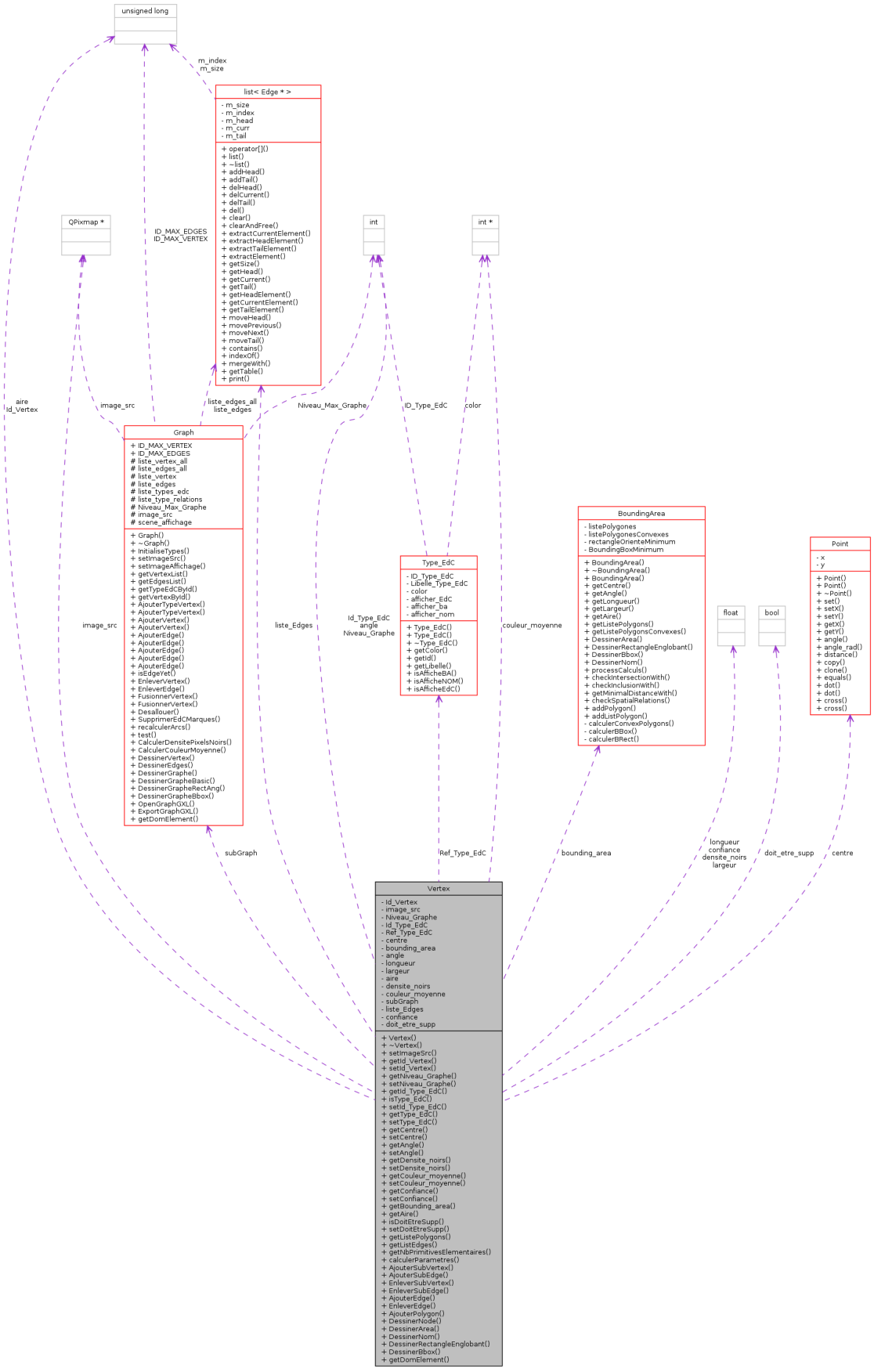


FIG. 4.4 – Modélisation UML - Classe Vertex

La classe vertex comprend aussi un entier nommé Niveau_Graphe auquel on affecte la hauteur du noeud. Cette valeur correspond aussi au nombre de fusions opérés sur ce vertice. Plus ce nombre est faible, et plus l'élément correspond à un niveau bas (quadrilatère ou vecteur par exemple).

Cette classe contient enfin toutes les caractéristiques statistiques de l'EdC comme la densité de pixels noirs, la couleur... ainsi que les propriétés géométriques de l'élément (longueur/largeur de l'approximation, aire...)

4.3 Edge (Arc)

La classe Edge représente quant-à-elle la liaison orientée entre deux Vertex.

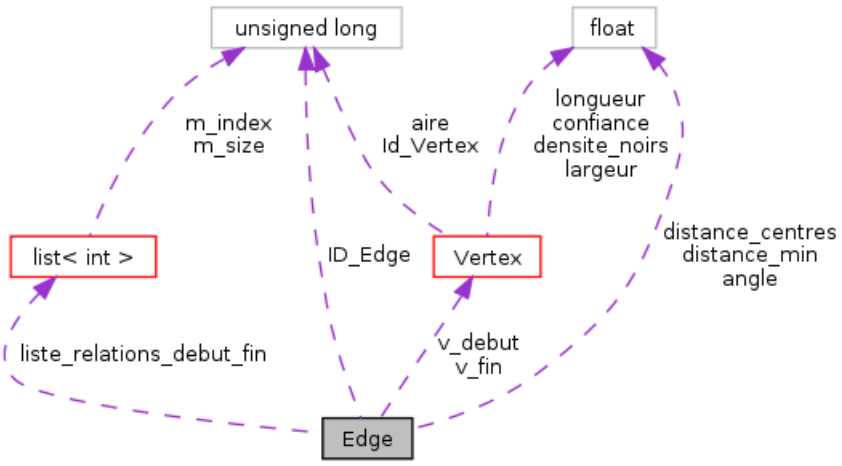


FIG. 4.5 – Diagramme de classe - Edge

V_debut et v_fin sont respectivement les pointeurs vers les vertex de départ et d'arrivée de l'arc. Les valuations sur cet arc sont de trois types :

- la distance entre les deux centroïdes des vertex
- la distance minimale entre les EdCs
- l'angle entre les deux EdCs
- deux listes de relations

Remarques :
La liste des relations contient les identifiants de la classe Type_Relation décrite dans la suite de ce chapitre. Cette modélisation nous permet ainsi de ne garder qu'une seule instance de l'arc pour plusieurs relations spatiales, et évite donc la duplication d'infor-

mations (minimiser le nombre d'arcs pour l'algorithme de recherche/comparaison de graphes). Nous pouvons de plus aisément ajouter de nouveaux types de relations non prévues (dans la classe Type_Relation).

La version actuelle du programme ne manipule que des arcs orientés et ne fait appel qu'à une seule liste de relations. C'est donc à l'algorithme de recherche/validation d'arcs que revient le rôle de considérer les inférences (si l'on cherche une relation EST_A_DROITE_DE, alors il faudra considérer aussi les arcs EST_A_GAUCHE_DE d'orientation inverse).

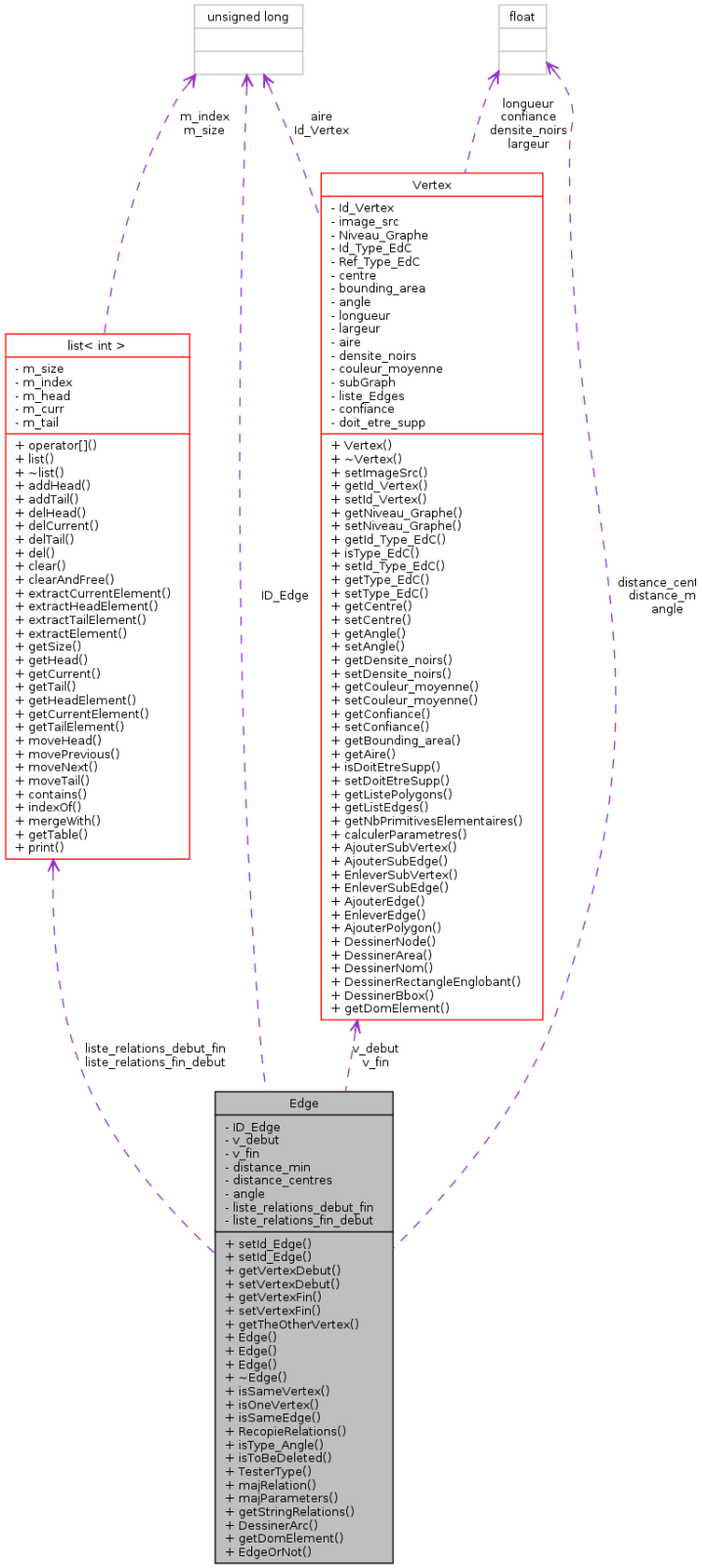


FIG. 4.6 – Modélisation UML - Classe Edge

Enfin, la classe **Edge** contient la fonction de plus haut niveau de détection des relations de voisinage.

4.4 BoundingArea

BoundingArea est une classe contenant l'ensembles des polygones définissant l'élément de contenu, ainsi que les trois approximations (Bounding Box, Bounding Rectangle ainsi que l'enveloppe convexe).

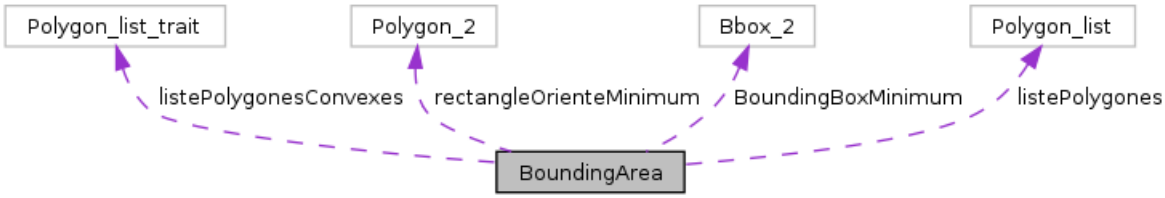


FIG. 4.7 – Diagramme de Classe - BoundingArea

Dans le cas d'un élément seulement défini par un ensemble de points est ajouté, un mesh sera réalisé en vue d'obtenir un ensemble de polygones. La figure suivante détaille plus spécifiquement les attributs et méthodes de la classe.

FIG. 4.8 – Modélisation UML - Classe BoundingArea

La classe BoundingArea donne accès aux méthodes suivantes :

– checkInclusionWith : précise sur un EdC donné en paramètre est inclus dans l'EdC courant (utilisation des rectangles englobants orientés)
– checkIntersectionWith : précise si les rectangles englobants des deux éléments s'intersectent

4.5 Type d'élément de contenu

Le type d'élément de contenu est implémenté sous la forme d'une classe possédant un identifiant unique, un libellé, une couleur, ainsi que plusieurs booléens permettant de préciser si l'on souhaite afficher tous les EdCs de ce type, si l'on désire que les EdCs soient affichés sous leur forme approximée ou en affichant simplement leur nom.

La liste esinitialementnt composée des éléments connus suivant :
– 0. Indéfini
– 1. Vecteur
– 2. Quadrilatère
– 3. Composante Connexe
– $n > 3$ autres, définis par l'utilisateur.

Cette classe pourra contenir la liste des règles de description, ou une référence vers un graphe décrivant le type général de l'élément.

Pour afficher des informations sur le graphe, une syntaxe a été créée :
– %a → show the angle
– %i → show the ID
– %W → show the width
– %L → show the length
– %n → show the # of sub-elements
– %N → show the graph level
– %A → show the Area
– %x → show the x position
– %y → show the y posistion
– %t → show the trust

Par exemple, pour afficher la position des quadrilatères, insérez ceci dans le type Quadrilatère : Quad (%x ;%y).

Voici l'interface de dialogue donnat accès à ces informations :

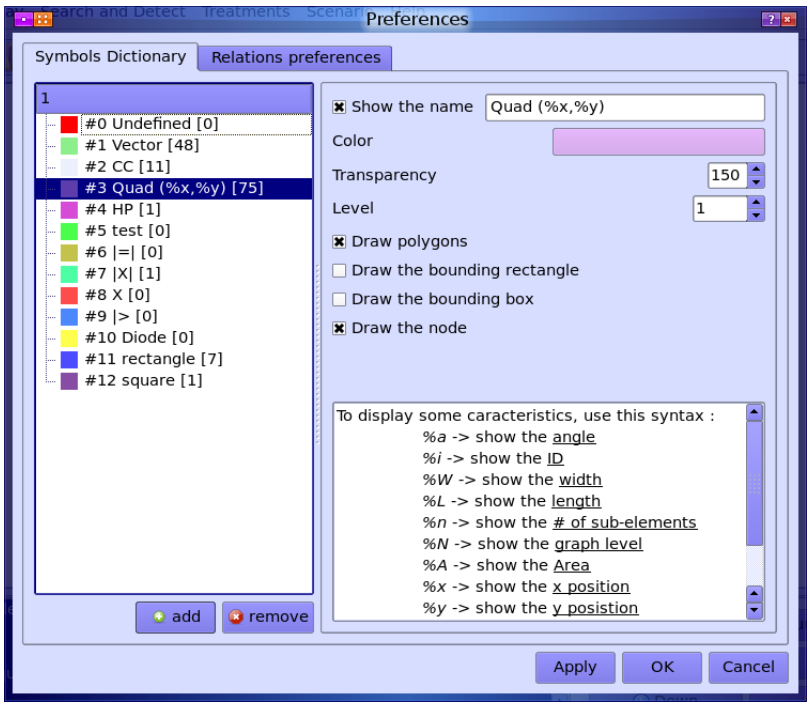


FIG. 4.9 – Préférences d'affichage

Il est de plus possible de modifier ces valeurs dans les fichiers XML (scénarios et fichier sGXL).

4.6 Type de relation

La classe contenant les différents types de relations inter-EdCs est définie par deux champs : un identifiant unique ID, et une chaîne de caractère qui sera affichée sur l'image résultante, en milieu de parcours de l'arc. Les relations connues à ce jours sont les suivantes :

- 0. INDEFINI
- 1. ANGLE_EN_L
- 2. ANGLE_EN_l
- 3. ANGLE_EN_T
- 4. ANGLE_EN_t
- 5. ANGLE_EN_P
- 6. ANGLE_EN_X
- 7. ANGLE_EN_s
- 8. ANGLE_EN__
- 9. EST_A_GAUCHE

- 10. EST_A_DROITE
- 11. EST_AU_DESSUS
- 12. EST_EN_DESSOUS
- 13. EST_TOTALEMENT_INCLU
- 14. CONTIENT_TOTALEMENT
- 15. CHEVAUCHEMENT

D'autres éventuelles relations pourrons y être ajouté simplement. Les relations 1 à 8 ne sont pour l'instant pas recalculées, il s'agit de relations fournies par le programme de vectorisation externe que nous employons.

L'interface suivante illustre l'ajout d'une relation entre deux éléments :

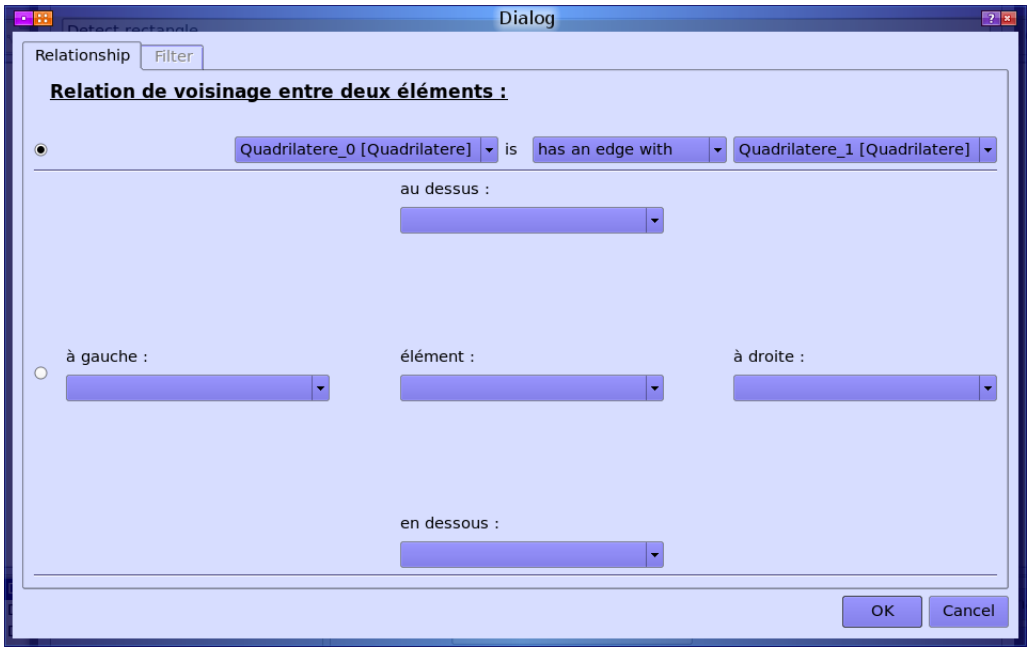


Fig. 4.10 – Ajout d'une relation

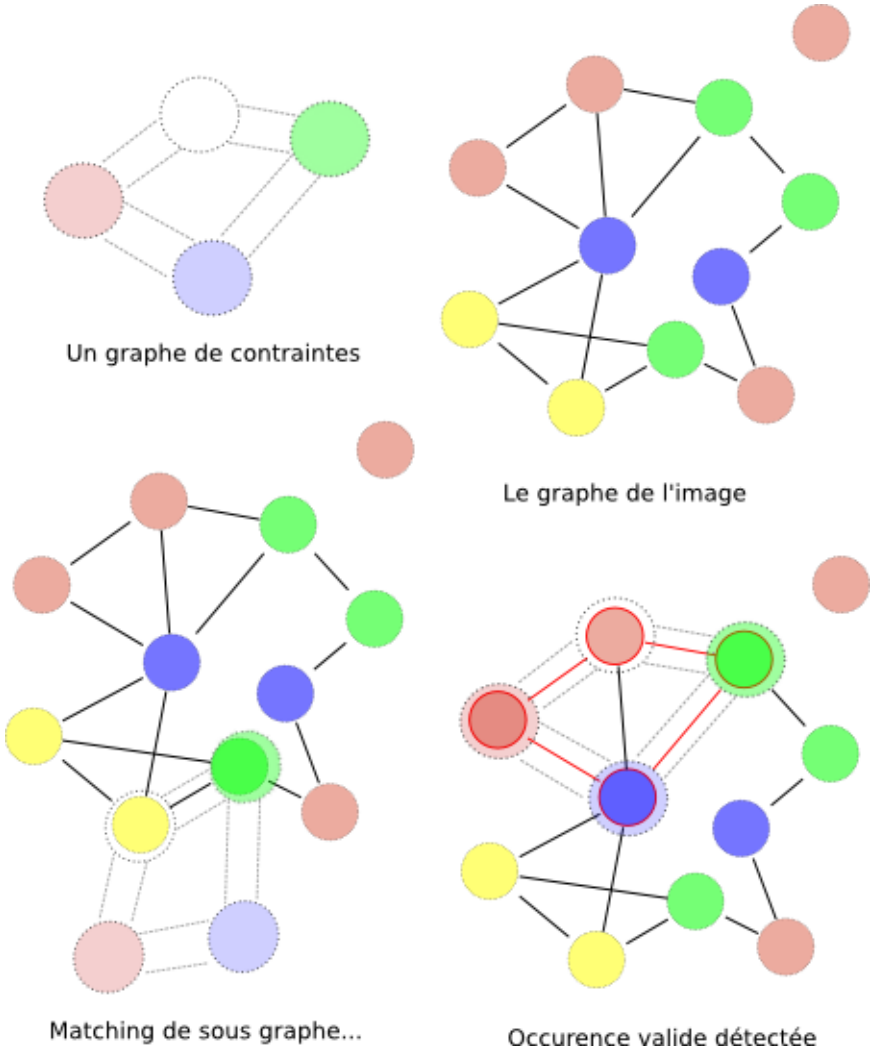
Chapitre 5

Algorithme de recherche de sous graphe minimal

Pour détecter et localiser des symboles dans des images, nous avons choisis d'utiliser un algorithme d'isomorphisme de sous graphes.

5.1 Définition de la recherche

Il s'agit ici de rechercher les occurences d'un graphe de référence (le symbole, définit au préalable) dans un autre graphe, le graphe de l'image.
À la manière des stencils, l'idée est de tenter d'appareiller le graphe symbole dans l'immensité du graphe de l'image requête et de ne retenir que les instances valides, comme le montre la figure suivante.



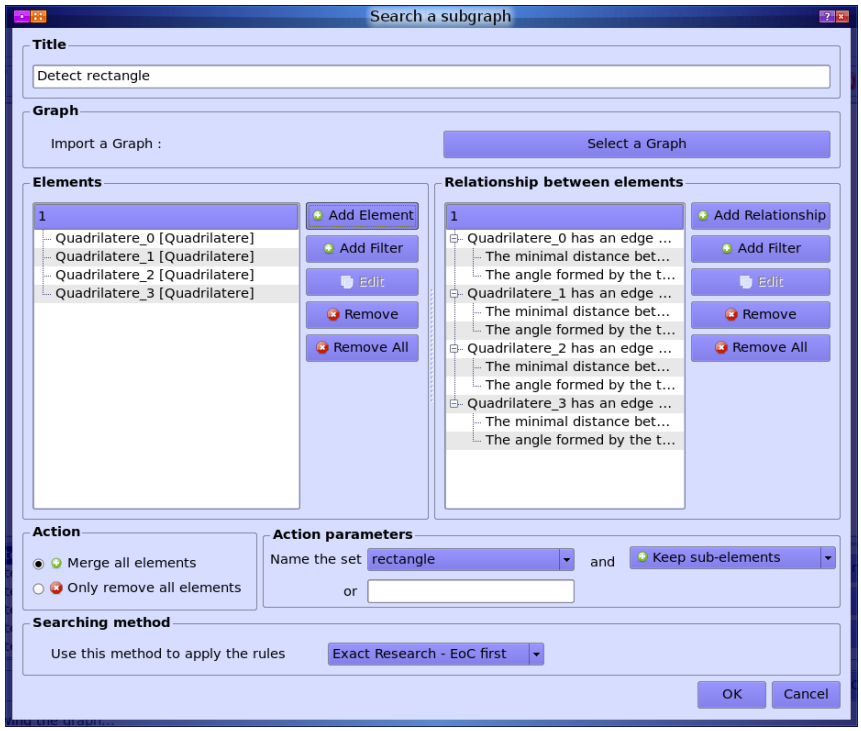
En plus du matching de graphes, des contraintes sont ajoutées aux noeuds et aux arcs afin de restreindre le champ de recherche sur des symboles précis. Chaque contrainte sera défini par l'objet contraint, ainsi que part son intervalle de définition (minimal et maximal).

Les contraintes noeuds sont de plusieurs catégories : les contraintes topologiques (type d'éléments de contenu, nombre de sous éléments constituant l'élément recherché), les contraintes géographiques (position par rapport au document) et géométriques (aire, longueur et largeur estimées, densité de pixels noirs...)

Les contraintes arcs portent quant à elles sur les relations de voisinage. Elles sont de deux types : topologiques (type de relation, sens) et géométriques (angle et distances minimale et inter-centroïdes entre deux éléments de contenu).

5.2 Interface de recherche de sous graphe

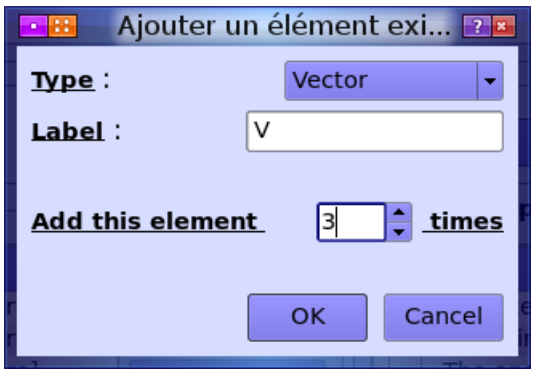
La définition des symboles à rechercher se réalise, pour l'instant, à l'aide d'une interface graphique. C'est donc à l'utilisateur de décrire le graphe recherché ainsi que les contraintes permettant de discriminer les bons symboles.



Nous retrouvons les éléments suivants :

- Le titre de la recherche : la chaîne de caractères qui sera affiché dans le scénario
- Une importation d'un fichier graphe (sgxl, gxl) qui récupère le graphe contenu dans le fichier.
- La définition du graphe, avec à gauche les noeuds et leurs contraintes, à droites les arcs et leurs contraintes inter-EdC. Chaque Noeud du graphe à un nom unique ce qui permet de parfaitement définir les arcs et leur orientation.
- La zone d'action : Une fois que l'algorithme a trouvé tous les sous graphes répondant aux critères, une action est effectuée. Soit nous souhaitons supprimer les éléments du sous graphe, soit nous voulons créer un nouvel élément en fusionnant les éléments.
- Le choix de l'algorithme à employer.

Pour ajouter un nouveau noeud au graphe, l'interface suivante a été conçue (choix du type, du nom et du nombre d'ajout de ce type à réaliser) :



Ensuite, pour ajouter une contrainte à un noeud, il suffit de sélectionner un noeud et de fire appel à cette boîte de dialogue :

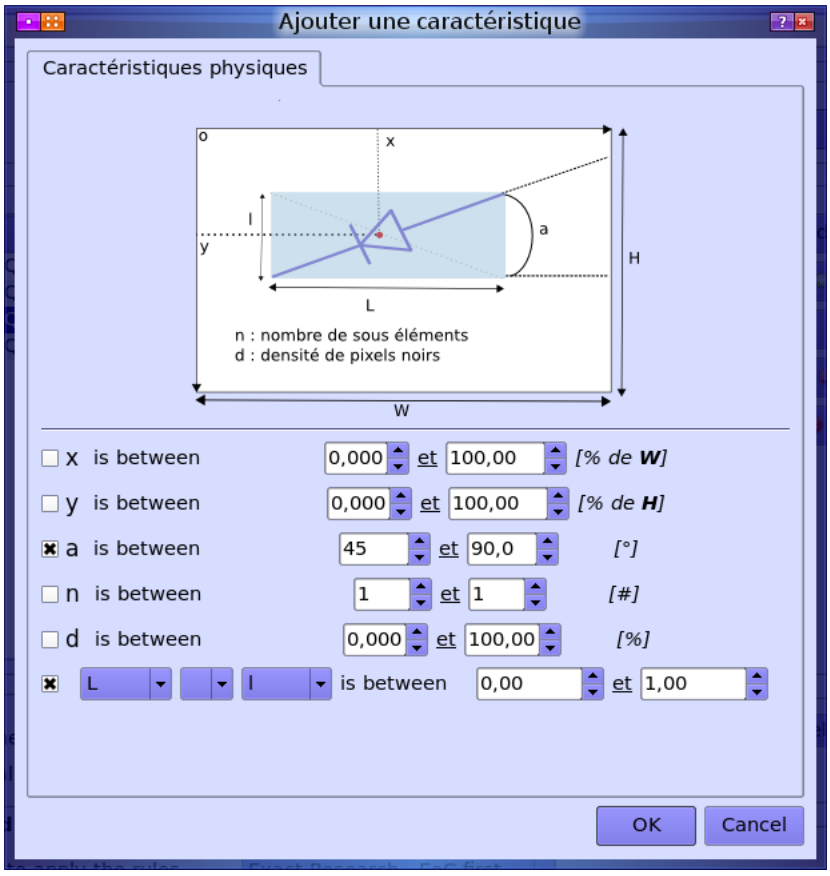


FIG. 5.1 – Ajout d'une contrainte noeud

La recherche d'un symbole doit donc répondre à plusieurs attentes : la recherche de sous graphes minimaux validant le graphe de référence du symbole, ainsi que la validation de chaque contraintes noeuds et arcs.

Le problème de recherche de sous graphes (Subgraph matching) est connu appartenir à la classe de problèmes NP-complet, il n'existe donc pas d'algorithme le résolvant en temps polynomial. Il est courant dès lors d'employer une algorithme de type Branch and Bound (PSE), auquel nous avons intégré une validation de contraintes ainsi que plusieurs heuristiques de parcours et d'élimination de solutions.

5.3 Algorithme

L'algorithme PSE que nous proposons est implémenté en c++ et est basé sur un parcours récursif du graphe. Voici les spécifications de la fonction **SubGraphResearchPSE** :

Nom : SubGraphResearchPSE
Entrée : Graph (Vertex, Edges) : G ,GraphFilter (VertexFilter, EdgeFilter) : GF
Sortie : booléen SubGraphFound, Liste de sous graphes validés LSG
Préconditions : LSG est vide mais alloué , VertexFilter non affectés
Postconditions : LSG = ensemble des sous graphes de G validants GF
Documentation : Retourne la liste des sous graphes validant les contraintes noeuds et contraintes arcs de GF.

Algorithme :
SubGraphResearchPSE(G,GF,LSG)

Algorithme 2 Algorithme de recherche de sous graphes

Entrées: un Graph G, un GraphFilter
Sorties: Liste de sous graphes validés

```
Soit SGVF le sous graphe formé par les VertexFilter déjà affectés
Soit SGV le sous graphe formé par les Vertex affectés de SGVF
Si il reste des VertexFilter non encore affectés Faire Alors
  Si vide(SGVF) = vrai Alors
    Choisir un VertexFilter VF parmis GF
  Sinon
    Choisir un VertexFilter VF non encore affecté et tel qu'il existe au moins un arc
    entre VF et SGVF
  Fin Si
  Soit SEF, l'ensemble des EdgeFilter de VF dont les VertexFilters suivants appar-
  tiennent à SGVF
  Pour tout vertex v appartenant au voisinage de SGV Faire
    Si v valide VF Alors
      contraintesArcsValidees ← vrai
      Pour tout EdgeFilter ef de SEF Faire
        Si il n'existe pas d'arc entre v et le vertice affecté du 2eme VertexFilter de
        ef validant ef Alors
          contraintesArcsValidees ← faux
        Fin Si
      Fin Pour
      Si contraintesArcsValidees = vrai Alors
        affecter(VF,V)
        SubGraphResearchPSE(G,GF,LSG)
        //on est revenu, donc on enleve v de VF
        desaffecter(VF,v)
      Fin Si
    Fin Si
  Fin Pour
  Retourner faux
Fin Si
Soit SGVF le sous graphe formé par les VertexFilter déjà affectés
// fin du parcours du graphe
Si SGV est une solution et n'est pas dans LSG Alors
  ajouter(LSG,SGV)
Fin Si
//pour parcourir tout le graphe
Retourner faux
```

Pour simplifier les explication, prenons l'exemple suivant. Il s'agit de deux étapes successives de l'algorithme de matching.

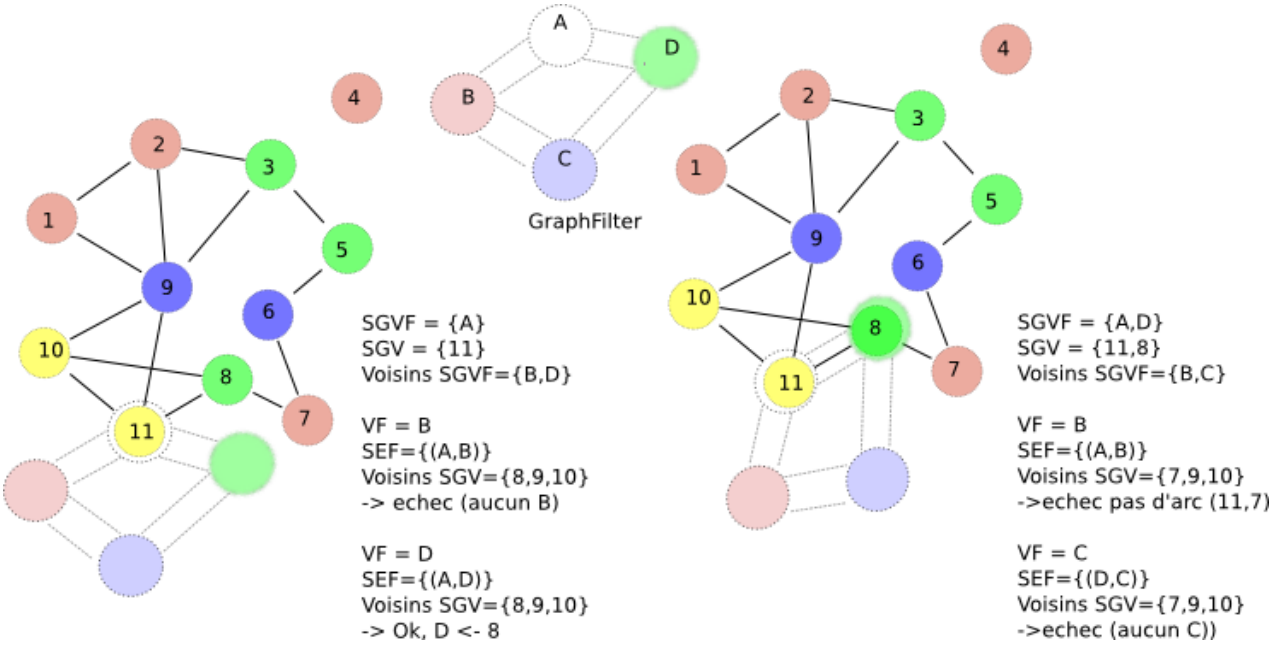


FIG. 5.2 – Étapes 1 et 2 de l'algorithme de recherche

Pour être simple, nous commençons par choisir un noeud du graphe symbole (GraphFilter) à tester (le A). Ensuite, nous parcourons le graphe de l'image pour trouver un noeud validant les contraintes noeuds (dans l'exemple, considérons que le noeud 11 valide le noeud A du symbole). Nous choisisons ensuite un des noeuds voisins de A non encore validé (B ou D, par exemple B) et nous regardons s'il existe un voisin de 11, validant B (même couleur). Il n'en existe pas, on passe donc au D. Il existe un noeud (8), voisin de 11 et qui valide D, on vérifie les contraintes arcs ($(11, 8)$ et (A, D)). Tests réussi, on affect donc D à 8et on passe à l'étape suivante.
On choisit un noeud du symbole non encore affecté (parmi B et C) et on se rend compte que ni B, ni C n'ont de noeud potentiel dans le voisinage de (11 et 8)...échec... On passe donc au noeud suivant, etc...

5.4 Heuristiques de choix des noeuds et arcs contraints à explorer

Le parcours des noeuds à explorer ne se fait pas aléatoirement, mais repose sur une heuristique qui considère les propriétés suivantes :

Prendre en premier lieu les noeuds dont le type recherché est "rare".

→ ajouter une proportionnelle au type de l'élément à trouver, en fonction du nombre d'éléments de ce type présents dans l'image.

Plus les paramètres du noeud sont contraints et plus le nombre de vertex répondant aux critères diminue.

→ ajouter une proportionnelle au nombre de contraintes à satisfaire, ainsi qu'à la taille de la borne de valeurs.

Il en va bien entendu de même avec les contraintres arcs, il est préférable de choisir en premier l'arc qui contient le plus de contraintes fortes (dont la fenêtre de valeurs min/max est réduite).

5.5 Propriétés de la méthode - résultats

D'après la conception de l'algorithme, nous optenons ici une recherche exhaustive, toutes les solutions, tous les symboles validant les contraintes sont détectés. Il s'agit de plus d'une recherche exacte : un sous graphe de l'image valide ou non l'ensemble des contraintes et, bien que l'on ait intégré des domaines de valeurs dans les contraintes numériques, nous ne pouvons obtenir des graphes approximatifs, validant par exemple 70 % de la requête.

Pour intégrer ce type de données, nous devons ajouter un taux d'appartenance (validation) sur les contraintes structurelles du graphe recherché. Il ne s'agirait donc plus d'une recherche de sous graphe minimal, mais d'une recherche approximative. (Ce champ de recherche nécessite plus de travaux !)

Néanmoins, le recours aux contraintes sur les domaines (minimum et maximum) et non sur des valeurs nous permet d'être relativement général lors de la description des symboles recherchés. Nous obtenons de bon résultats par exemple lors de la détection de rectangles, de carrés ou de triangles, et ce, en utilisant seulement les angles inter-EdC. Il faut de plus remarquer que si l'on utilise seulement les angles inter-EdC comme contraintes arcs, la détection possède alors une invariances à la rotation et au facteur d'échelle.

Une manière astucieuse pour détecter des symboles, est de décomposer celui-ci en éléments simples. Par exemple, trouver un rectangle est très facile, il suffit de définir 4 noeuds de type Quadrilatère, et de leur affecter une relation de voisinage 2 à 2 (1 avec 2, 2 avec 3, 3 avec 4 et 4 avec 1) et d'ajouter une contrainte sur l'angle inter-EdC

(entre 88 et 92). Ensuite, pour détecter un carré, il suffit de détecter les rectangles dont le rapport Longueur sur largeur est entre 0.95 et 1.0 !

En construisant ainsi petit à petit, mais de manière sûre l'information, nous sommes capable de détecter des éléments beaucoup plus complexes que si nous décrivons directement le graphe du symbole.

Voici un petit exemple illustrant ces propos. Supposons que nous désirons trouver toues les occurences des symboles carrés avec les diagonales.

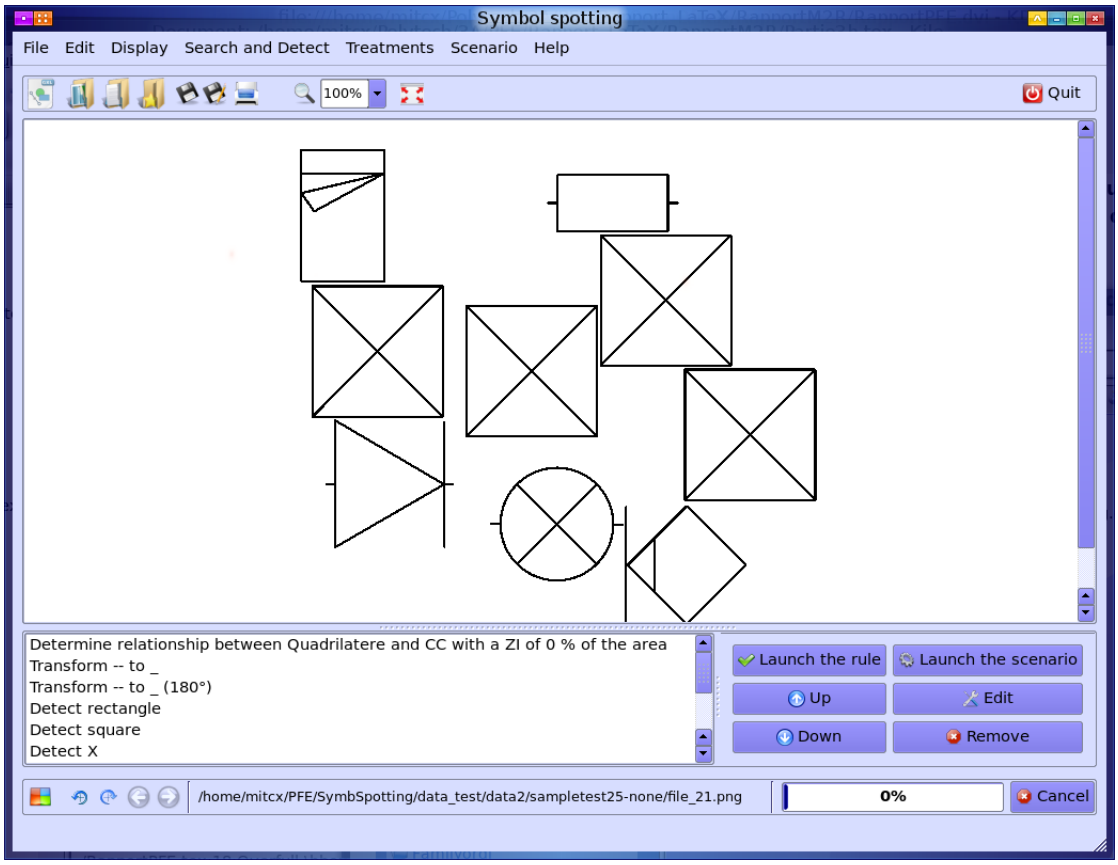


FIG. 5.3 – Image de départ

Commençons par construire le graphe initial (vectorisation) :

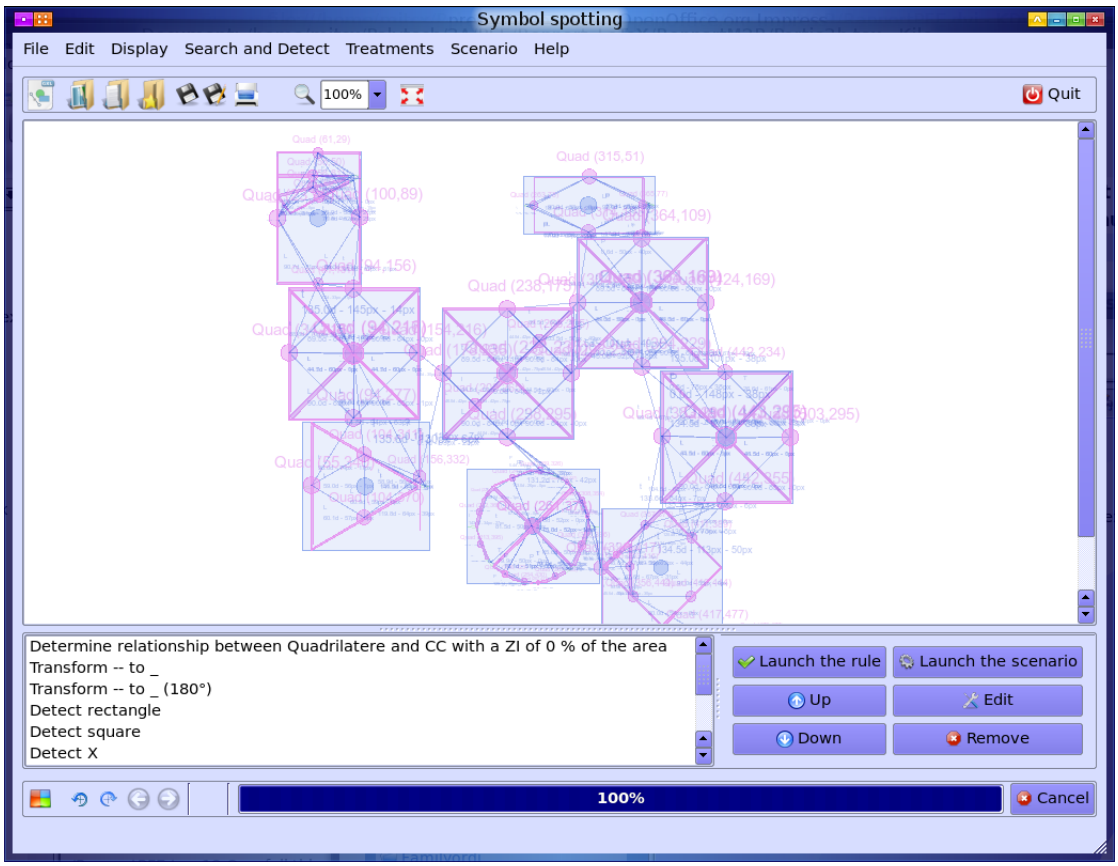


FIG. 5.4 – Graphe délivré par une vectorisation

Appliquons maintenant la règle décrire précédemment de détection des rectangles.

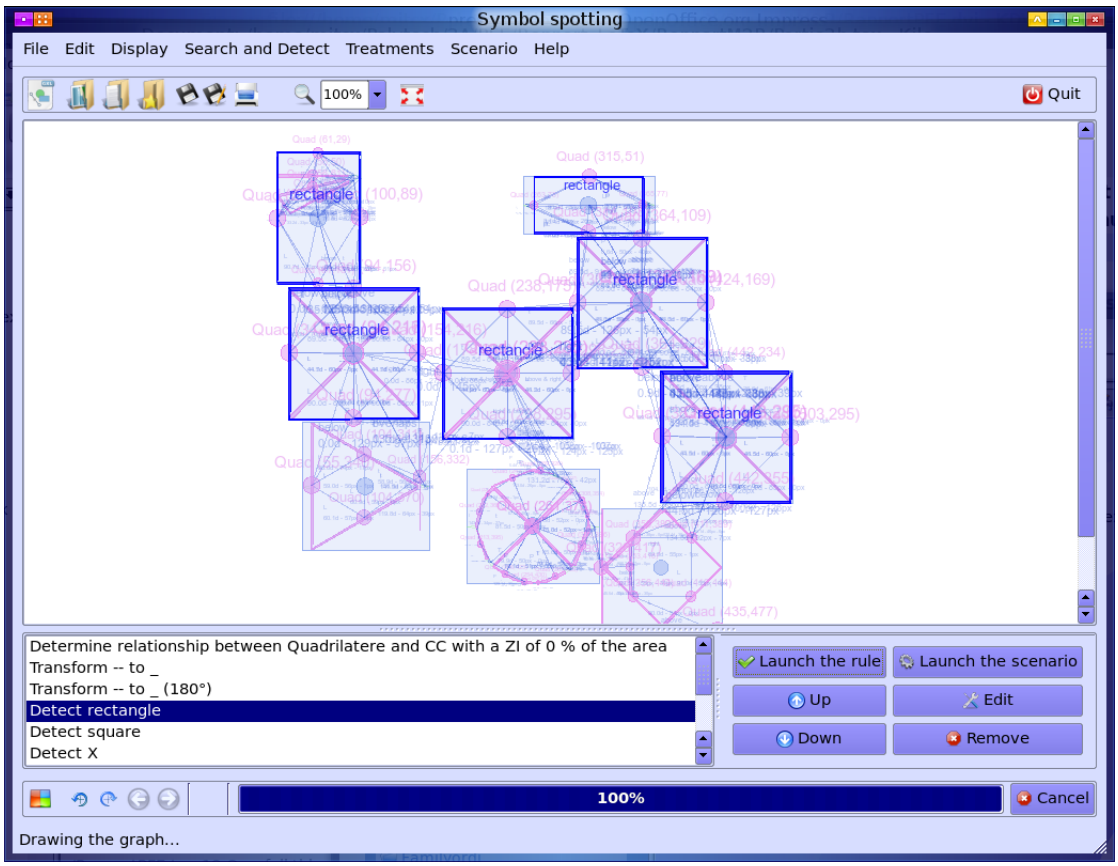


FIG. 5.5 – Détection des rectangles

Ensuite, nous pouvons rechercher les carrés (rectangles dont la longueur et la largeur sont similaires).

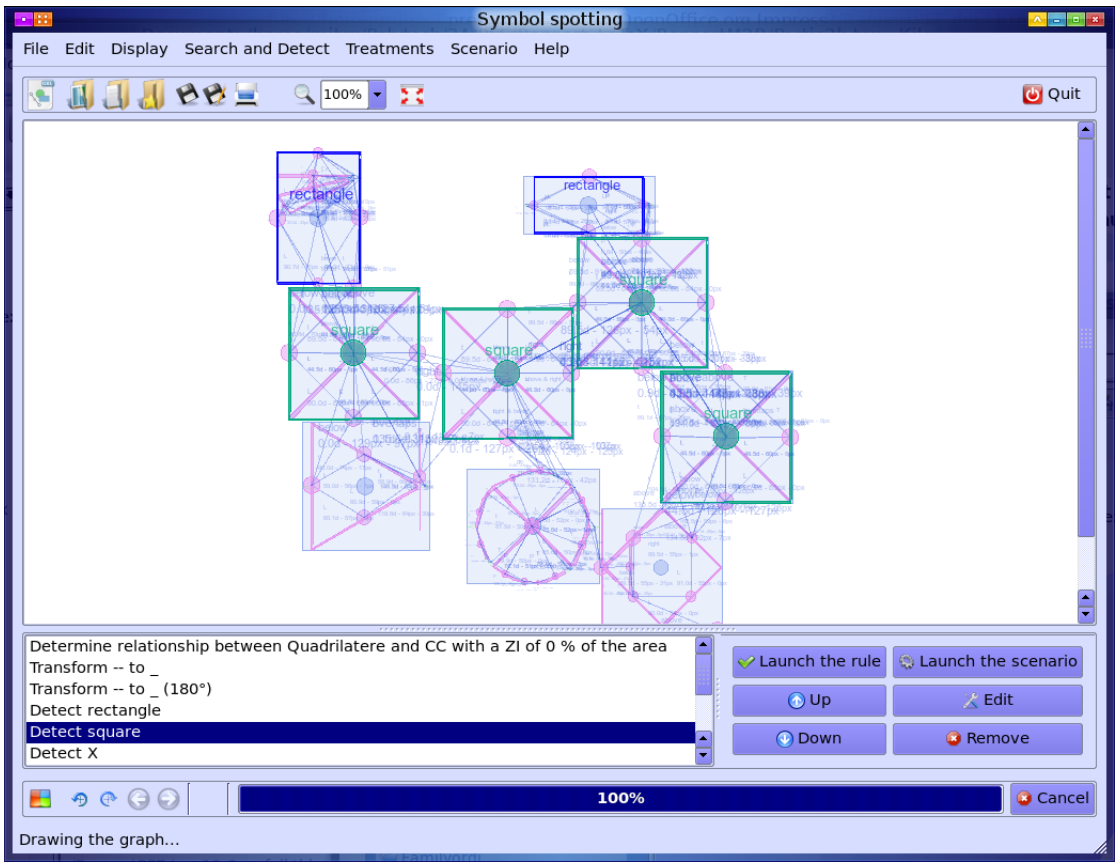


FIG. 5.6 – Détection des carrés

Un fois que nous avons les carrés, détectons les croix (deux quadrilatères dont l'angle entre les deux est entre 88 et 92 et dont la distance entre leur centre est proche de 0).

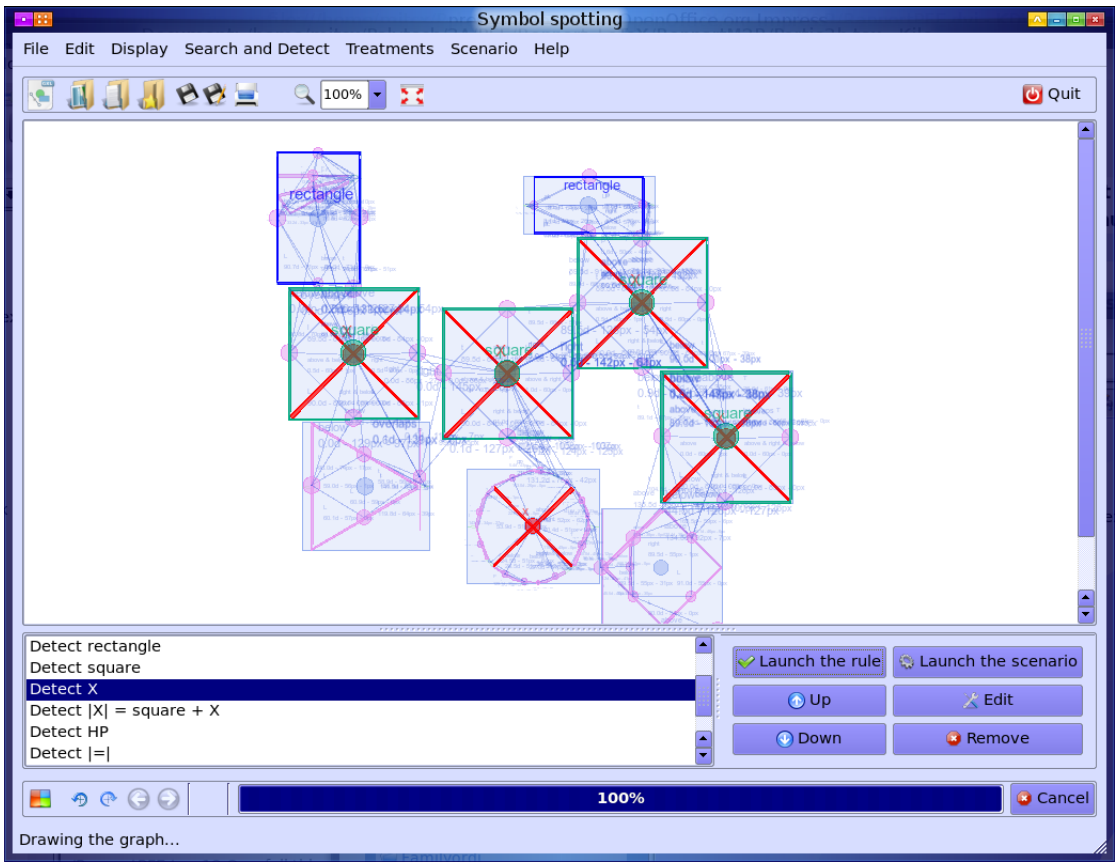


FIG. 5.7 – Détection des croix X

Nous obtenons bien toutes les croix ! Il suffit donc de chercher toutes les croix entièrement inclus dans un carré pour localiser notre symbole de départ.

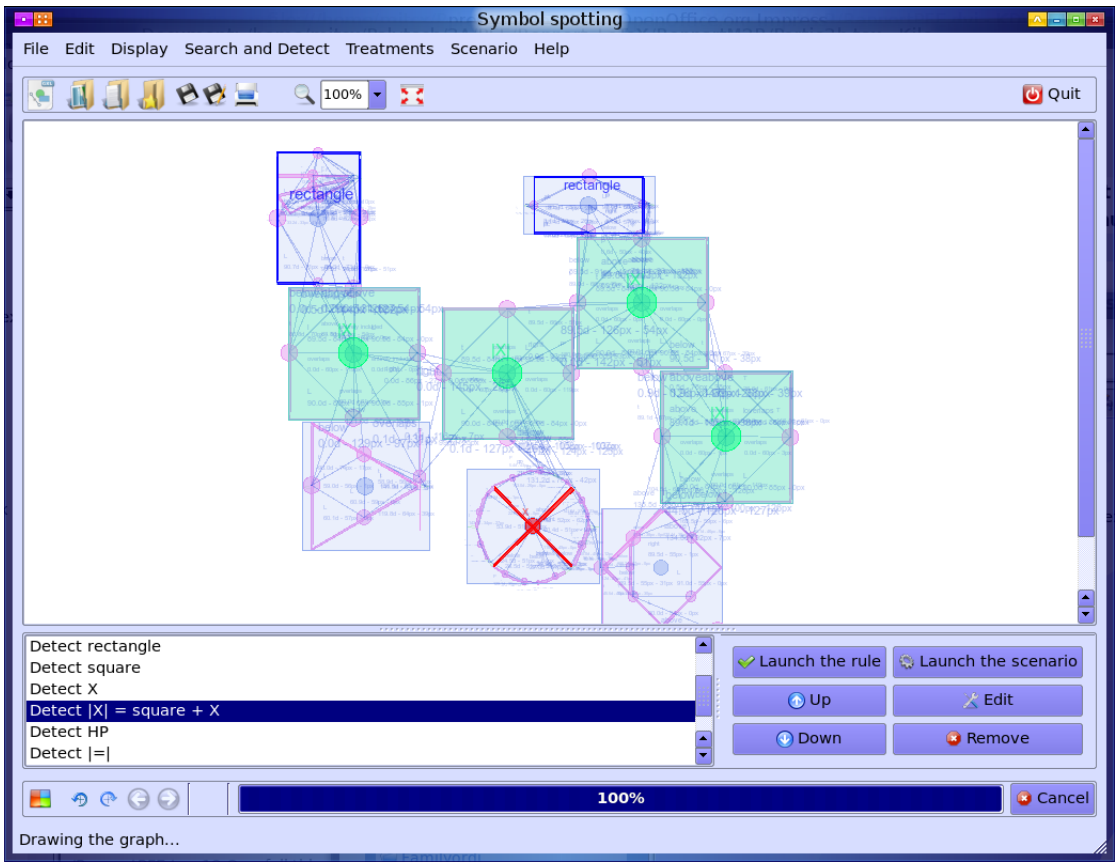


FIG. 5.8 – Détection des symboles [X]

L'opération a fonctionné, toutes les boîtes ont été localisées !

Plusieurs remarques sont à faire cependant :

Cette méthode de recherche est fortement dépendante de l'étape de vectorisation. Si cette opération est stable par rapport au bruit ou aux facteurs d'échelle ou de rotation, alors la recherche sera elle aussi efficace.

L'efficacité de la méthode dépend aussi beaucoup des règles de description employées pour localiser les symboles. C'est à l'utilisateur d'employer de bonnes règles, assez discriminentes pour ne détecter que les symboles désirés, mais assez générales pour ne pas être affecté par de petites variations.

De plus, il s'agit ici d'une recherche exacte. Cette recherche est déjà longue, mais sera d'autant plus couteuse en temps de calcul que le nombre d'éléments (noeuds et arcs) du graphe de l'image requête sera important.

Enfin il est nécessaire de préciser que certains symboles sont plus difficiles à détecter. Notamment les symboles constitués d'arcs ou de cercles ou présentant des courbures

d'une manière générale. De part l'utilisation de polygônes comme éléments géométriques primaires constituant les EdCs plus complexes, les courbures sont complexes à manipuler. La résolution de ce problème passe soit par la reconnaissance d'arcs lors de l'étape de vectorisation, soit par l'élaboration des règles permettant de détecter des courbures. Après quelques essais, la deuxième proposition semble possible mais relativement coûteuse et dépendante de l'angle de courbure.

Chapitre 6

Modules de gestion et de traitement

Ce chapitre décrit les différents modules principaux utilisés par le programme.

6.1 MainManager

La classe MainManager contient l'ensemble des éléments suivant :

- La liste des éléments de contenu (type de noeuds) connus
- La liste des relations connues par le programme (type d'arc)
- L'image courante
- Le graphe associé à l'image
- Le scénario
- Un pointeur vers l'interface principale du logiciel

Voici la diagramme de classes de la classe MainManager.

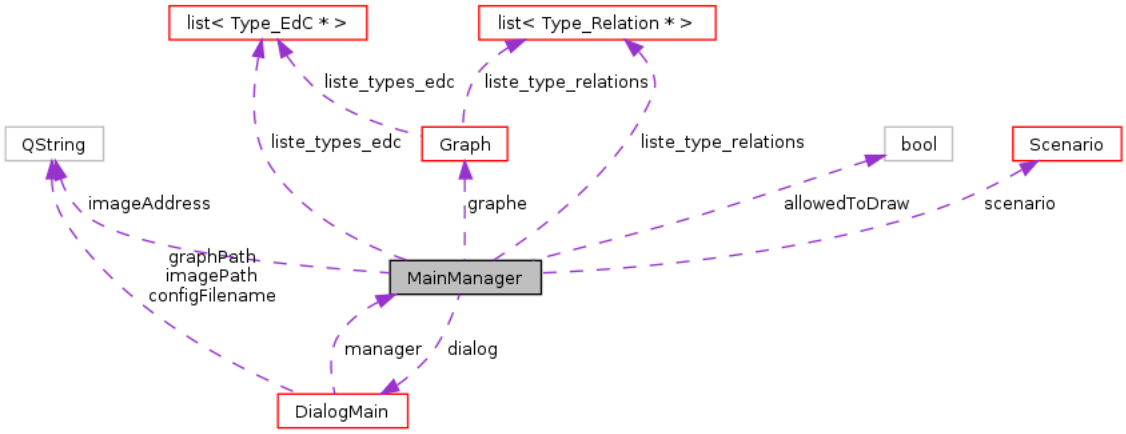


FIG. 6.1 – Diagramme de classes - classe MainManager

Cette classe gère les actions suivantes :

– Appel à l'affichage du graphe
– Import/Export du scenario
– Lancement de la vectorisation
– Ouverture des graphes (.dat)

La gestion des ouvertures des fichiers GXL, SGXL et SVG s'opère dans la classe Graph.

6.2 DialogMain

DialogMain est l'interface principale du programme et implémente de nombreuses méthodes comme notamment l'appel aux autres boîtes de dialogue, etc...

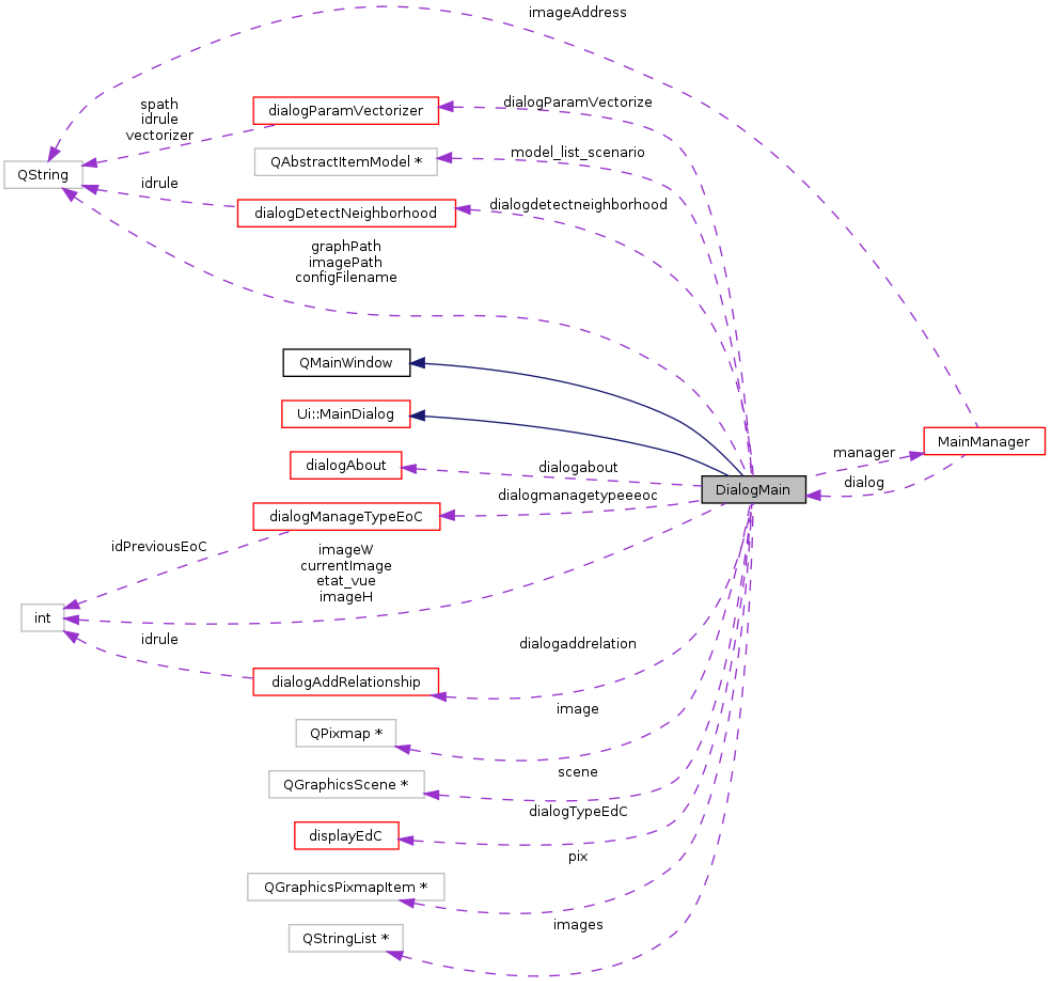


FIG. 6.2 – Diagramme de classes - classe DialogMain

La création d'une nouvelle interface de dialogue implique de spécifier l'instance dans

la définition de la classe DialogMain, puis de faire appel à la méthode .show() pour l'afficher.

6.3 Scénarii et règles d'action

Afin de décrire des règles de traitement ou de reconnaissance de symboles, nous avons intégré un module de gestion de règles sous forme de scénarii.

Un scénario est composé de diverses informations. Il y a notamment la liste des Types d'éléments de contenus connus par le programme. Il contient de plus les règles de traitement et d'action à opérer. L'ensemble des types d'EdC manipulés dans les règles doivent être connus par le scénario.

Pour simplifier le codage des règles, nous avons choisis d'utiliser l'héritage de classe. La classe Regle suivante doit donc être héritée par toute nouvelle règle.

6.3.1 La classe Regle

La classe Regle gère donc l'ensemble des informations nécéssaires à la manipulation de règles. Elle contient les champs suivants :

- Type : une chaîne de caractères unique identifiant le type de règle
- ID : une chaîne de caractères unique identifiant la règle dans un scénario
- Description : une chaîne de caractères affichée dans l'interface des scénarii.

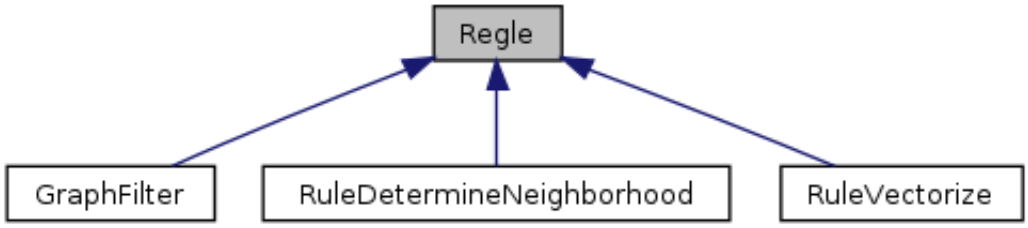
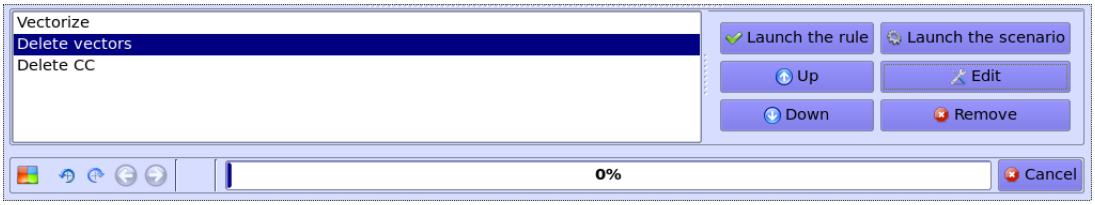


FIG. 6.3 – Héritages de la classe Regle

Cette classe contient de plus 4 fonctions virtuelles. La propriété virtuelle (ou abstraite en JAVA) signifie qui est nécessaire de définir ces fonctions lors de chaque héritage. Ces 4 fonctions servent à l'import et l'export en XML, ainsi que l'application de la règle. La dernière méthode doit changer certains types d'EdC en type reconnus. Cette fonction est appelées lorsqu'il y a un conflit entre les types d'EdC du scénario et ceux du graphe courant.

6.3.2 Interface de gestion des scenarii

La gestion des scénarii se fait pour l'utilisateur au travers de la fenêtre principale du programme. Chaque ligne représente une règle.



Les 6 boutons situés à droites gèrent les règles (Monter/Descendre/Editer/Supprimer/Appliquer une règle et Appliquer le scénario).

6.3.3 Format XML des scenarii .sce

Comme nous l'avons précisé antérieurement, les scénarii contiennent deux listes, les types d'éléments de contenu ainsi que les règles de traitement.

Une exemple de scénario est présent en **annexe A**.

6.3.4 Creer une nouvelle regle

1. Creer une classe RuleXXX qui hérite de la classe Regle (publique)
2. Déclarer et implémenter les méthodes abstraites runThisRule, ImporterRegle, ExporterRegle et RemplacerEoCTypesIDs dans la nouvelle classe RuleXXX.
3. Ajouter dans l'énumération de la classe Type_Regle (.h) un nouveau type TypeXXX, dans le .cpp ajouter un const QString sTypeXXX("XXX") ("XXX" doite être unique!) et inserer celui-ci dans le listeTypesRegles et dans listeTypesReglesGraphTreatment ou listeTypesReglesImageTreatment (sans oublier d'incrémenter les tailles des tableaux.
4. Pour tous les constructeurs de la classe RuleXXX, faire appel à la méthode : Regle(Type_Regle : :getXMLToken(Type_Regle : :TypeXXX)) qui permet d'initialiser les informations communes des règles.

La classe est désormais connue et identifiée. Il ne reste plus qu'à creer l'interface des paramètres de la regle et d'instancier la classe RuleXXX. L'ajout de la règle au scenario ce fait très simplement par l'intermédiaire de la fontion AjouterRegle (scenario-¿AjouterRegle(RuleXXX)). L'ajout, l'édition et la suppression des règles se réalisent dans le fichier DialogMain.cpp.

6.4 Structure des fichiers .sgxl

La structure des fichiers GXL étant un peu trop restrictive (DTD), noius avons décidé d'utiliser un version plus souple : le (sgxl) Simplified GXL.

Chaque noeud du graphe (node) contient une liste de formes géométriques primaires reconnues par le logiciel (GeometricShapes). En général il s'agit de polygônes.

Un exemple de fichier sgxl constitue l'**annexe B**.

Chapitre 7

Le logiciel SymbSpot

Ce dernier chapitre est consacré à la présentation du logiciel que nous avons conçu lors du Projet de Fin d'Étude. Nous détaillerons les caractéristiques techniques du logiciel, comment nous l'avons implémenté, comment nous avons testé notre code, quelles sont les fonctionnalités courantes et futures du logiciel, etc...

7.1 Caractéristiques techniques

Le logiciel était au départ une amélioration d'un programme développé antérieurement. Ce logiciel utilisait une technologie assez ancienne qui ne favorisait pas un codage simple, rapide et optimal des algorithmes définis dans les chapitres précédents.

7.1.1 Librairie d'interfaces

Après avoir chercher plusieurs librairies réalisant des interfaces, nous avons choisis d'utiliser la librairie Qt4 de la société Trolltech.

Cette bibliothèque graphique fournit de multiples éléments d'interfaçage comme des boutons, des champs de données, ainsi qu'un ensemble de classes évoluées permettant de créer des interfaces de dialogues, des zones de dessin (graphe de scène 2D), un accès 3D openGL, une gestion aisée des threads et des mutex, un module XML, un module de gestion de protocole réseau, etc... Bref tout un ensemble de fonctions de haut niveau très intéressantes.

Fig. 7.1 – Trolltech

Cette librairie possède de plus l'avantage d'être parfaitement portable sous une ma-

jorité de systèmes d'exploitation comme Linux, Windows ou MacOS.

7.1.2 Langage de programmation et gestionnaire de graphes/listes

Le choix du langage de programmation s'est imposé de lui même. Le C++ est un langage simple, performant et surtout rapide d'exécution. Puisque le projet utilise une représentation sous forme de graphe, il est nécessaire de prendre un langage qui manipule très rapidement des graphes, et le c++ est justement très bien adapté.
Il existe de plus plusieurs librairies de gestion de graphes en c++, comme notamment la bibliothèque BOOST, mais nous avions commencé à développer notre modélisation à l'aide d'une template modifiée de listes, et nous n'avons pas recoder l'ensemble. Ceci nous donne cependant l'avantage d'avoir un accès direct au code et donc d'optimiser notre implémentation aux contraintes du projet.

7.1.3 Librairie de calculs graphiques

Une fois la phase de conception réalisée, nous avons pu avoir un aperçu des caractéristiques que l'on désirait calculer, avec notamment la distance minimale entre deux éléments de contenu ou l'analyse en composantes principales. La plupart de ces calculs ont déjà été implémentés en langage c++. Étant donné que le sujet porte sur des éléments géométriques 2D, mais aussi pour être simple tout en restant assez général, nous avons entrepris d'utiliser la librairie de calculs géométriques CGAL.

FIG. 7.2 – Bibliothèque CGAL

La bibliothèque CGAL implémente de nombreuses fonctions et autres éléments géométriques, comme notamment des polygones, des rectangles, des droites, des segments, ou des ensembles de points etc... Concernant le calcul des propriétés géométriques, voici une liste non exhaustive des fonctionnalités de la librairie :

- calcul de l'enveloppe convexe d'un ensemble de points en 2D, 3D ou dD
- partitionnement 2D en polygones
- triangulation de Delaunay 2D et 3D
- diagrammes de Voronoï
- génération de mesh 3D
- recherche des k voisins les plus proches
- recherche par intervalles, dans un espace dD (2D, 3D...), par range and trees

- calcul des Bounding Volumes
- calcul des distances optimales
- ACP
- Structures kinetiques
- et surtout un Linear and Quadratic Programming Solver (QP Solver)

Cette librairie est vraiment une très bonne bibliothèque de fonctions ! L'utilisation et l'appel aux méthodes CGAL reste relativement simple pour peu que l'on soit habitué aux templates c++.

7.1.4 Tests unitaires et vérifications de l'efficacité du code

Pour vérifier la qualité du code que nous avons écris, nous avons eu recours à la méthode des tests unitaires. Ces tests sont simples : nous avons injecter des données virtuelles dont les résultats étaient connus (par le calcul) pour chaque algorithme codé. De plus, afin de simuler le parcours de l'ensemble des domaines de chaque valeur d'entrée (position des points des polygones, ...), nous avons appelé chaque fonction avec des paramètres tirés aléatoirement (et respectant les préconditions de la fonction bien entendu).

7.2 Fonctionnalités du logiciel

Une fois la phase de conception terminée, nous avons entrepris la réalisation physique du logiciel. Voici un aperçu du logiciel :

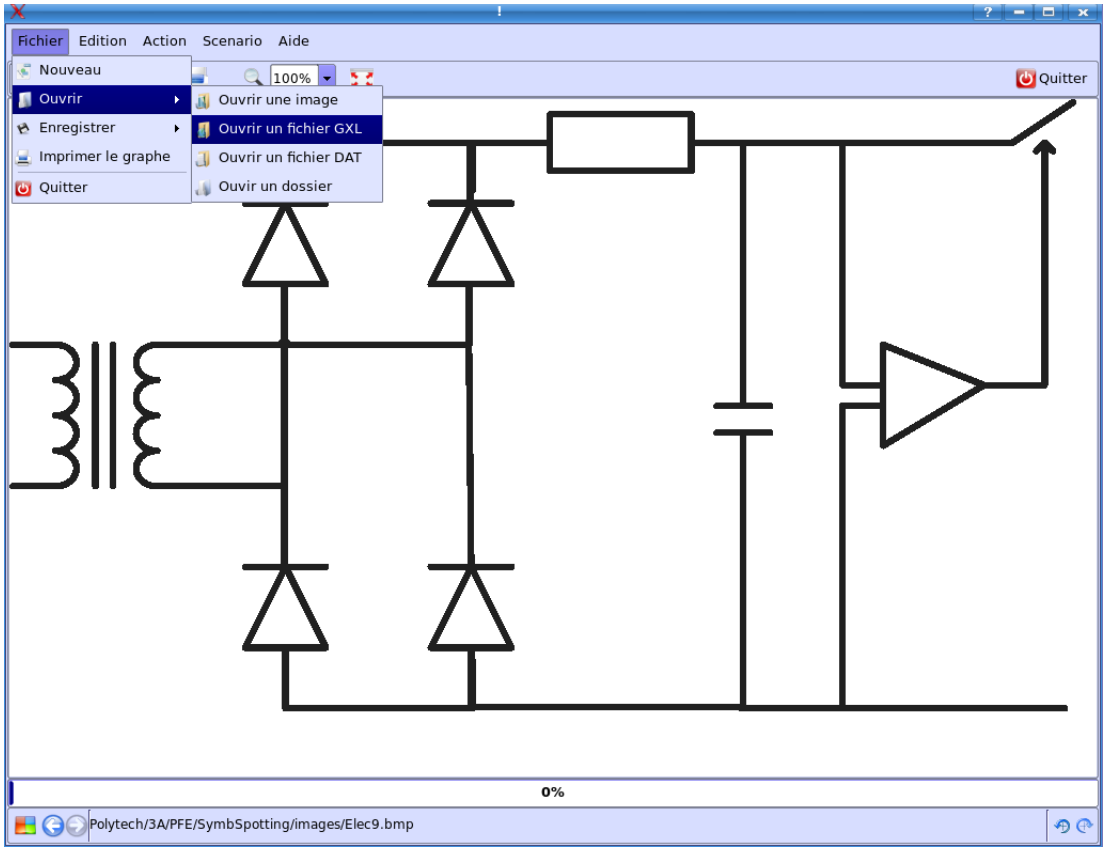


FIG. 7.3 – Menu d'ouverture du logiciel

Nous retrouvons un menu permettant d'ouvrir différents types d'images ainsi que les fichiers de graphes GXL et DAT. L'onglet Action permet de lancer une binarisation de l'image (non encore implémentée), la vectorisation de l'image en éléments de base ainsi que des actions primaires de traitement (temporaires).

La zone de visualisation contenant l'affichage du résultat est un graphe de scène 2D. Ceci nous permet d'afficher des éléments non pas directement sur l'image mais dans graphe, afin d'être aisément modifiable (translation, export svg, ...).

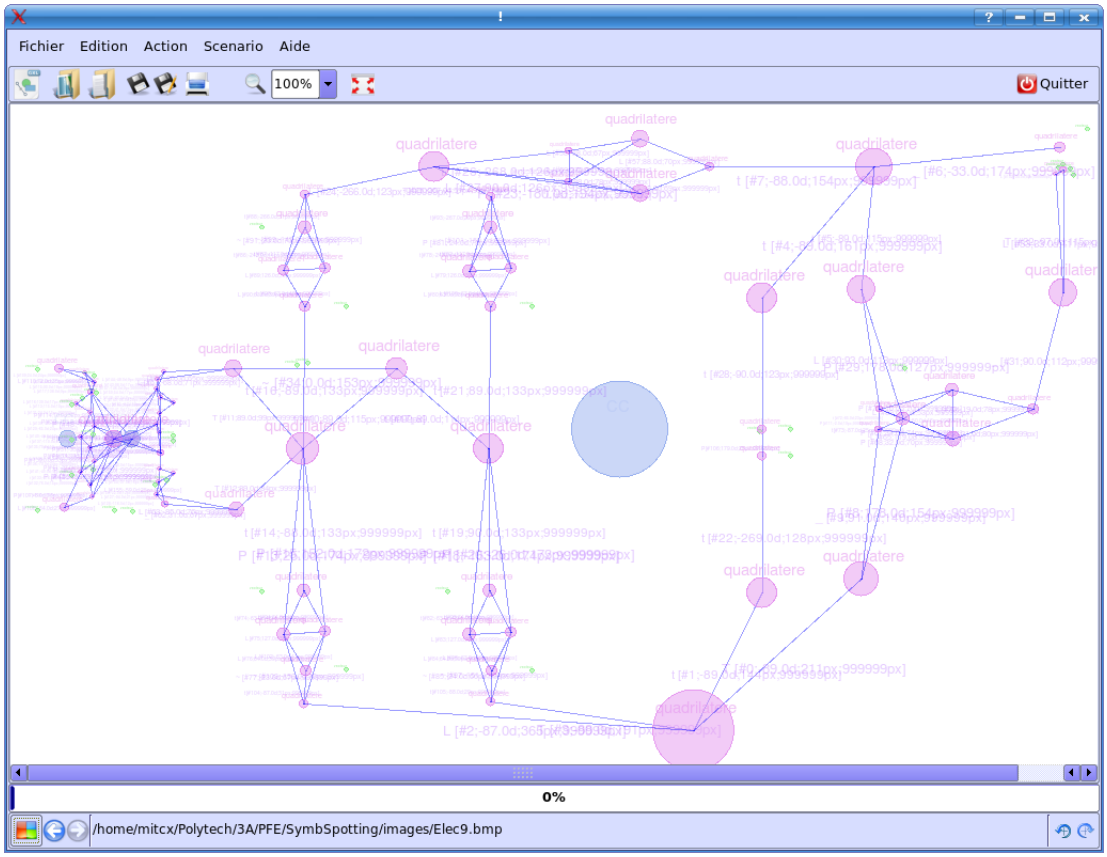


FIG. 7.4 – Affichage du graphe

Dans la figure précédente, nous retrouvons le graphe du schéma électrique que nous avons vectorisé. Les noeuds sont bien entendu modélisés par des cercles de couleur et les arcs par des segments bleus. Les étiquettes sont aussi ajoutées au dessus des noeuds et les informations contenus sur les arcs sont aussi intégrés. Le rayon des cercles est proportionnel au volume de l'élément qu'il représente et les couleurs sont propres au type d'EdC modélisé.

Il est de plus possible de choisir d'autres vues du graphes en cliquant sur le bouton situé en bas à gauche de la fenêtre principale. Plusieurs vues sont possibles :

1. mode choix de l'utilisateur : le graphe affiché correspond aux options de chaque type d'élément de contenu (boîte de dialogue)
2. mode image seule
3. mode image et graphe simplifié (cercles & arcs)
4. mode image et polygones
5. mode rectangles englobants

6. mode bounding box

Le figure suivante illustre la vue image et graphe simplifie.

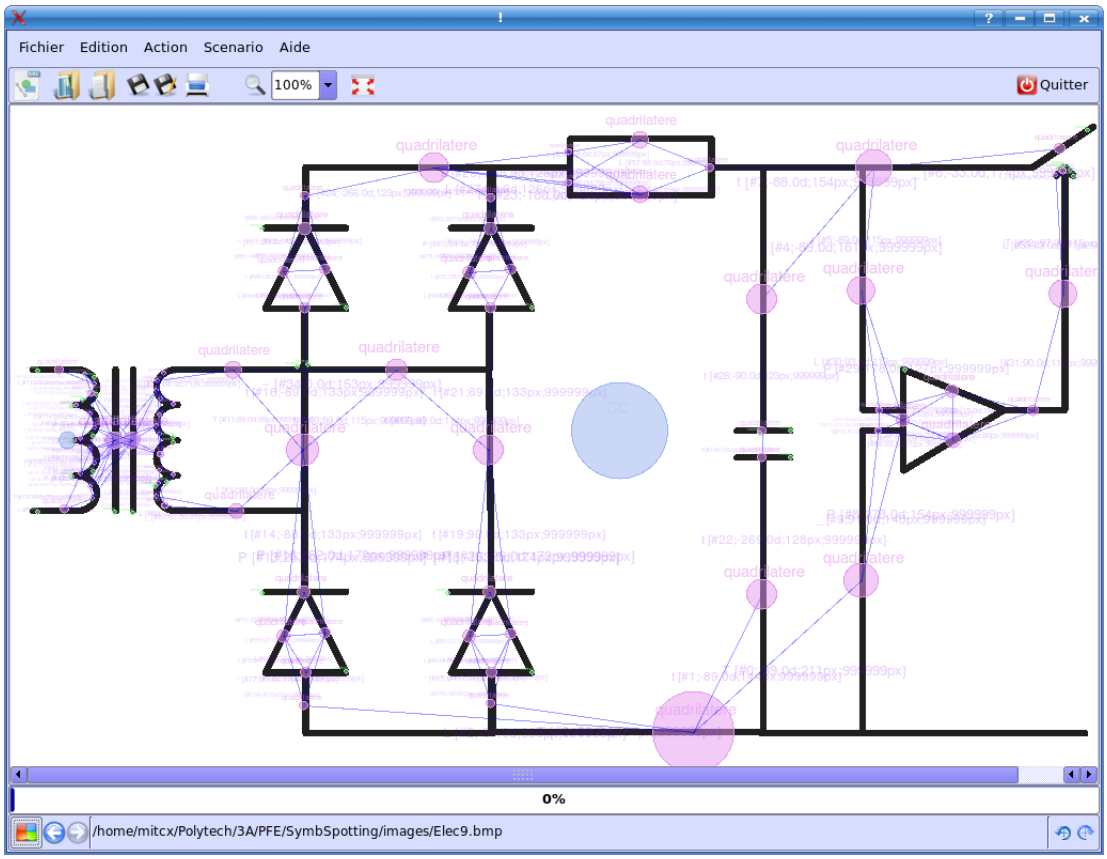


FIG. 7.5 – Affichage du graphe sur l'image

L'une des grandes particularité de Qt est que l'affichage est en vectoriel, ce qui permet d'avoir un zoom de grande qualité.

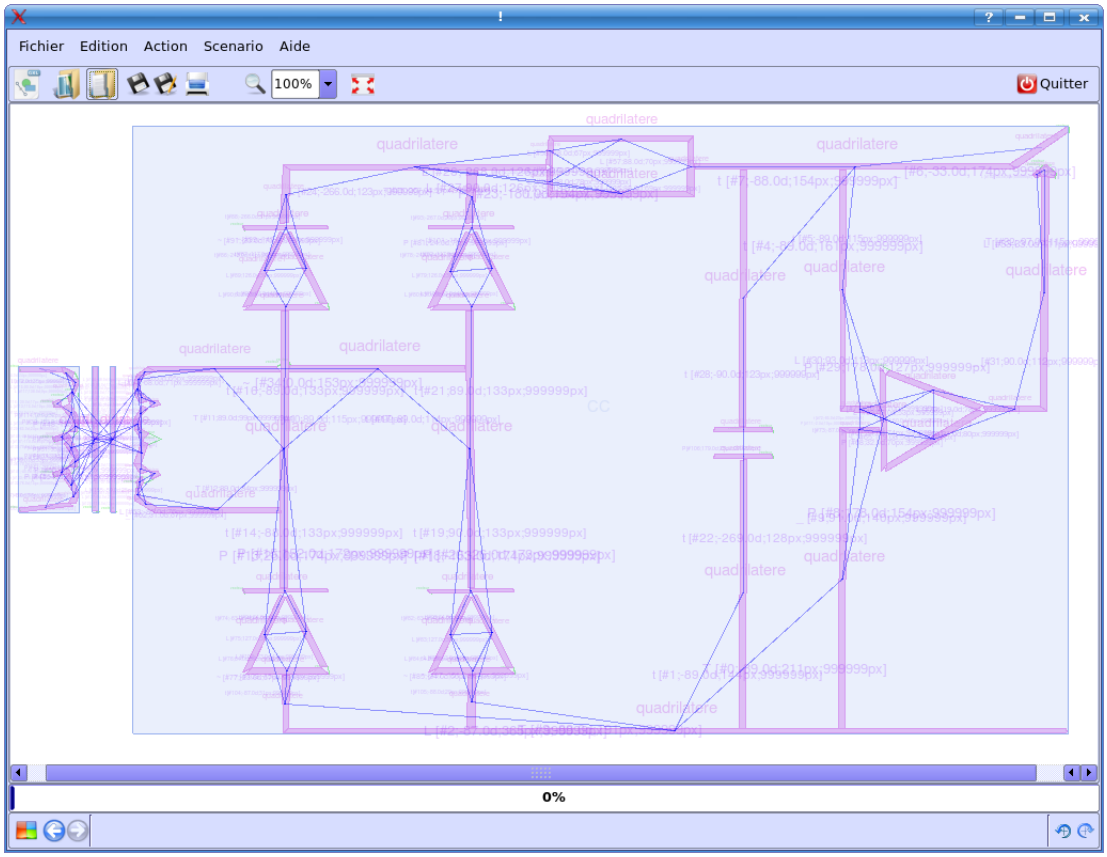


FIG. 7.6 – Affichage des EdCs sous leur forme polygonale

Ici nous observons chaque élément sous leur forme polygonale, on reconnaît donc immédiatement les éléments déterminés par la vectorisation.

7.3 Évolutions et améliorations

Le logiciel que nous avons conçu est à ce jour une version relativement aboutie. Il présente de nombreuses fonctionnalités notamment dans la gestion de graphe 2D (affichage, import/export, manipulation, recherche...), et peut dès lors très bien être utilisé comme plateforme de traitement/manipulation de graphes.

Voici cependant quelques pistes à explorer pour améliorer le logiciel ainsi que la détection de symboles.

– Concevoir un algorithme d'apprentissage de graphe : en présentant plusieurs images du même objet, le logiciel calcule les graphes de chacun et trouve les

similarités (graphe minimal) avec des invariances choisies pas l'utilisateur (ex : l'orientation)
- Ajouter/Définir de nouvelles caractéristiques (statistiques, géométriques ou géographiques) : par exemple les moments de Zernike [5] ou autre...
- Améliorer la détection de voisinage (plus rapide, plus efficace)
- Améliorer l'algorithme de recherche de sous graphes pour éliminer rapidement les zones inintéressantes.
- Tester d'autres algorithmes de recherche avec incertitude.
- Stocker et gerer différentes primitives géométriques de base (liste de points puis pavage, courbe de Bezier pour les courbures)
- Porter l'application à la librairie CGAL 3.3 et activer le calcul des distances minimales réelles (et non des estimations)

Conclusion

Ce projet de fin d'étude a porter sur le problème du symbol spotting, et principalement sur la modélisation d'images sous forme de graphes. J'ai notamment pu réaliser un graphe bi-niveaux composé de noeuds multi-valués (éléments de contenu) et d'arcs (représentant les relations spatiales). Ce modèle, assisté par les approximations géométriques des éléments rend possible une gestion aisée des différentes informations contenues dans le graphe.

La conception et implémentation du logiciel rend possible le parcours rapide et simplifié du graphe, mais cela ne replacera pas bien entendu la conception d'algorithmes de parcours efficaces pour les méthodes d'application des règles ou d'appariement de graphes.

Du point de vue ingénierie, ce projet m'a apporté une expérience assez enrichissante puisque j'ai pu accomplir les phases essentielles de la conception d'un logiciel. J'ai de plus manipuler quelques algorithmes fort intéressants comme le calcul des enveloppes convexes, le calcul des rectangles englobants ou la détermination de la distance minimale entre deux polygones convexes.

Du point de vue recherche j'ai pu appronfondir mes connaissances sur plusieurs domaines parmi lesquels figures la recherche de symboles dans des documents, la recherche et l'appariement de graphes... J'ai de plus pu m'employer à tester un algorithme de type PSE possédant des contraintes sur les noeuds et sur les arcs.

Bibliographie

[1] Godfried T. Toussaint, *Solving geometric problems with the rotating calipers*, Proceedings of IEEE MELECON'83, Athens, Greece, May 1983

[2] Jean-Yves Ramel, *Propositions pour la représentation et l'analyse de documents numériques*, Habilitation à diriger des recherches, Tours, Novembre 2006

[3] Bernd Gärtner and Svend Schönherr. *An efficient, exact, and generic quadratic programming solver for geometric optimization.* In Proc. 16th Annu. ACM Sympos. Comput. Geom., pages 110-118, 2000.

[4] Delalandre Mathieu et al. *Reconnaissance de Symboles par Approche Structurelle Globale-Locale,Basée sur l'Utilisation de Scénarios, et Exploitant une Représentation XML des Données*

[5] Delalandre Mathieu et al. *A statistical and structural approach for symbol recognition, using XML modelling,SSPR, 2002*

[6] Delalandre Mathieu et al. *Analyse des documents graphiques : une approche par par reconstruction d'objets, juin 2007*

Annexe A : Structure XML des scenarii

```
<!DOCTYPE ScenarioDocument>
<Scenarii>
<EoCsTypes>
<TypeOfEoC showLabel="1" showElements="0" id="0"
showBoundingRectangle="0" showBoundingBox="0"
showNode="0" level="0" label="Undefined" >
<color G="0" R="255" A="255" B="0" />
</TypeOfEoC>
<TypeOfEoC showLabel="1" showElements="0" id="1"
showBoundingRectangle="1" showBoundingBox="0"
showNode="0" level="2" label="Vector" >
<color G="240" R="142" A="255" B="140" />
</TypeOfEoC>
<TypeOfEoC showLabel="1" showElements="1" id="2"
showBoundingRectangle="0" showBoundingBox="0"
showNode="0" level="-1" label="CC" >
<color G="176" R="152" A="50" B="239" />
</TypeOfEoC>
<TypeOfEoC showLabel="1" showElements="1" id="3"
showBoundingRectangle="0" showBoundingBox="0"
showNode="0" level="1" label="Quadrilatere" >
<color G="151" R="230" A="150" B="240" />
</TypeOfEoC>
<TypeOfEoC showLabel="1" showElements="1" id="4"
showBoundingRectangle="0" showBoundingBox="1"
showNode="1" level="10" label="hexagone" >
```

```
<color G="170" R="0" A="180" B="0" />
</TypeOfEoC>
<TypeOfEoC showLabel="1" showElements="1" id="5"
showBoundingRectangle="1" showBoundingBox="0"
showNode="1" level="10" label="octogone" >
<color G="255" R="255" A="180" B="0" />
</TypeOfEoC>
</EoCsTypes>
<scenario>
<rule type="selectionaction" id="rule0" >
<description>Supprimer les vecteurs</description>
<params>
<GraphFilter action="3" IDEoCMerge="0" searchingAlgorithm="0"
name="Supprimer les vecteurs" >
<EoCFilters typeEoC="1" id="Vector_0" name="Vector_0 [Vector]" />
</GraphFilter>
</params>
</rule>
<rule type="selectionaction" id="rule1" >
<description>Detecter les hexagones !</description>
<params>
<GraphFilter action="2" IDEoCMerge="4" searchingAlgorithm="0"
name="Detecter les hexagones !" >
<EoCFilters typeEoC="3" id="Quadrilatere_0"
name="Quadrilatere_0 [Quadrilatere]" />
<EoCFilters typeEoC="3" id="Quadrilatere_1"
name="Quadrilatere_1 [Quadrilatere]" />
<EoCFilters typeEoC="3" id="Quadrilatere_2"
name="Quadrilatere_2 [Quadrilatere]" />
<EoCFilters typeEoC="3" id="Quadrilatere_3"
name="Quadrilatere_3 [Quadrilatere]" />
<EoCFilters typeEoC="3" id="Quadrilatere_4"
name="Quadrilatere_4 [Quadrilatere]" />
<EoCFilters typeEoC="3" id="Quadrilatere_5"
name="Quadrilatere_5 [Quadrilatere]" />
<EdgeFilters typeRelationship="0" EoC1="Quadrilatere_0"
```

```
EoC2="Quadrilatere_1" name="Quadrilatere_0 has an edge with Quadrilatere_1" >
<filter equation="a" min="93" max="138" />
</EdgeFilters>
<EdgeFilters typeRelationship="0" EoC1="Quadrilatere_0"
EoC2="Quadrilatere_5" name="Quadrilatere_0 has an edge with Quadrilatere_5" >
<filter equation="a" min="93" max="138" />
</EdgeFilters>
<EdgeFilters typeRelationship="0" EoC1="Quadrilatere_1"
EoC2="Quadrilatere_2" name="Quadrilatere_1 has an edge with Quadrilatere_2" >
<filter equation="a" min="93" max="138" />
</EdgeFilters>
<EdgeFilters typeRelationship="0" EoC1="Quadrilatere_2"
EoC2="Quadrilatere_3" name="Quadrilatere_2 has an edge with Quadrilatere_3" >
<filter equation="a" min="93" max="138" />
</EdgeFilters>
<EdgeFilters typeRelationship="0" EoC1="Quadrilatere_3"
EoC2="Quadrilatere_4" name="Quadrilatere_3 has an edge with Quadrilatere_4" >
<filter equation="a" min="93" max="138" />
</EdgeFilters>
<EdgeFilters typeRelationship="0" EoC1="Quadrilatere_4"
EoC2="Quadrilatere_5" name="Quadrilatere_4 has an edge with Quadrilatere_5" >
<filter equation="a" min="93" max="138" />
</EdgeFilters>
</GraphFilter>
</params>
</rule>
</scenario>
</Scenarii>
```

Annexe B : Structure XML des graphes SGXL

```
 <!DOCTYPE sGXLdocument>
<Sgxl>
 <EoCsTypes>
  <TypeOfEoC showLabel="1" showElements="0" id="0"
  showBoundingRectangle="0" showBoundingBox="0"
  showNode="0" level="0" label="Undefined" >
   <color G="0" R="255" A="255" B="0" />
  </TypeOfEoC>
  <TypeOfEoC showLabel="1" showElements="0" id="1"
  showBoundingRectangle="1" showBoundingBox="0"
  showNode="0" level="2" label="Vector" >
   <color G="240" R="142" A="255" B="140" />
  </TypeOfEoC>
  <TypeOfEoC showLabel="1" showElements="1" id="2"
  showBoundingRectangle="0" showBoundingBox="0"
  showNode="0" level="-1" label="CC" >
   <color G="176" R="152" A="50" B="239" />
  </TypeOfEoC>
  <TypeOfEoC showLabel="1" showElements="1" id="3"
  showBoundingRectangle="0" showBoundingBox="0"
   showNode="0" level="1" label="Quadrilatere" >
   <color G="151" R="230" A="150" B="240" />
  </TypeOfEoC>
 </EoCsTypes>
 <Graphs>
  <graph NumberOfNodes="2" id="0" NumberOfEdges="1" >
```

```
   <node width="7" trust="1" graphlevel="0" id="node1"
   shape="3" density="0" area="1471" angle="1" lenght="208.038" >
    <center x="1539" y="1799" />
    <color G="0" R="0" B="0" />
    <GeometricShapes>
     <Polygon id="0" >
      <Point x="1436" y="1798" id="0" />
      <Point x="1440" y="1805" id="1" />
      <Point x="1644" y="1799" id="2" />
      <Point x="1643" y="1794" id="3" />
     </Polygon>
    </GeometricShapes>
   </node>
   <node width="193.127" trust="1" graphlevel="1" id="node2"
   shape="5" density="0" area="39541" angle="2" lenght="206.155" >
    <center x="1707" y="1299" />
    <color G="0" R="0" B="0" />
    <GeometricShapes>
     <Polygon id="0" >
      <Point x="1650" y="1206" id="0" />
      <Point x="1653" y="1213" id="1" />
      <Point x="1758" y="1209" id="2" />
      <Point x="1761" y="1202" id="3" />
     </Polygon>
    </GeometricShapes>
   </node>
   <edge distmin="0" distcenters="130.599" from="562"
   id="0" to="560" angle="91.9621" >
    <relation direction="->" type="4" id="0" />
   </edge>
  </graph>
 </Graphs>
</Sgxl>
```

Table des figures

1.1 Planning prévisionnel de développement 8

2.1 Modélisation d'un condensateur 13
2.2 Graphe de modélisation du condensateur 14

3.1 Approximations d'un élément de contenu EoC 20
3.2 Rotating Calipers 21
3.3 Rotating Calipers 22
3.4 Distance entre deux polygones 23
3.5 Angles entre deux EdCs, 2 cas possibles 24
3.6 Propositions de détection de voisinage 26
3.7 Occlusion par polygones - champ visuel 27
3.8 Occlusion par polygones - approximation 27
3.9 Détection des voisins - 8 connexité 28

4.1 Diagramme de Classe - Graph 30
4.2 Modélisation UML- classe Graph 31
4.3 Diagramme de classe - Vertex 33
4.4 Modélisation UML - Classe Vertex 34
4.5 Diagramme de classe - Edge 35
4.6 Modélisation UML - Classe Edge 37
4.7 Diagramme de Classe - BoundingArea 38
4.8 Modélisation UML - Classe BoundingArea 38
4.9 Préférences d'affichage 40
4.10 Ajout d'une relation 41

5.1 Ajout d'une contrainte noeud 45
5.2 Étapes 1 et 2 de l'algorithme de recherche 48
5.3 Image de départ 50

5.4 Graphe délivré par une vectorisation 51
5.5 Détection des rectangles 52
5.6 Détection des carrés 53
5.7 Détection des croix X 54
5.8 Détection des symboles [X] 55

6.1 Diagramme de classes - classe MainManager 57
6.2 Diagramme de classes - classe DialogMain 58
6.3 Héritages de la classe Regle 59

7.1 Trolltech 62
7.2 Bibliothèque CGAL 63
7.3 Menu d'ouverture du logiciel 65
7.4 Affichage du graphe 66
7.5 Affichage du graphe sur l'image 67
7.6 Affichage des EdCs sous leur forme polygonale 68

Résumé

Nous présentons dans ce document l'ensemble des démarches entreprises lors de l'élaboration du projet de fin d'étude. Le symbol spotting consiste à localiser des symboles connus dans des images. Nous abordons principalement une méthode basée sur la recherche et l'appariement de sous-graphes.

Mots clés

Reconnaissance de symboles, localisation d'éléments de contenu, graphe de voisinage, isomorphisme de sous graphe.

Abstract

In this report, we present the whole methodology we dealt with during our PFE on the symbol spotting technic. The symbol spotting method consist in the localisation of known symbols in document images. We describe our method to determine the symbols positions. Our branch and bound algorithm handle some successives treatments to recover symbols using a subgraph matching algorithm.

Keywords

Symbol spotting, document image analysis, graph modelling, subgraph isomorphism.

Encadrant :
M. Jean-Yves RAMEL
LI Tours
Equipe RFAI

Etudiant :
M. Julien MICHOT
DI 5ème année
Promotion 2004-2007