LA RESTRUCTURATION DES DOCUMENTS GRAPHIQUES DESTRUCTURÉS

Jacques Péré-Laperne

université de BORDEAUX

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR DE L'UNIVERSITÉ DE BORDEAUX

PRÉPARÉE À l'

ESTIA INSTITUTE OF TECHNOLOGY

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET INFORMATIQUE

Par Jacques Péré-Laperne

LA RESTRUCTURATION DES DOCUMENTS GRAPHIQUES DESTRUCTURÉS

Sous la direction de : Nadine Couture

Soutenue le 18/11/2019

Membres du jury :

Valérie Vigneras	Professeure, Bordeaux INP	Présidente
Rolf Ingold	Professeur, Université de Fribourg (Suisse)	Rapporteur
Salvatore Tabbone	Professeur, Université de Lorraine - LORIA	Rapporteur
Guy Mélançon	Professeur, Université de Bordeaux	Examinateur
Nicolas Schneider	Ingénieur de recherche AIRBUS	Examinateur
Nadine Couture	Professeure, Estia Institute of Technology	Directrice
Jean-Roch Guiresse	Docteur honoris causa, Université Wolverhampton	Invité

Résumé

Cette thèse traite de la restructuration des documents déstructurés de type PDF contenant des éléments graphiques tels que les schémas, les plans et les dessins, dans l'objectif de les restructurer. En nous appuyant sur la méthode KDD (Knowledge Discovery in Database) pour la restructuration des données, nous introduisons la méthode (A)KDD (Antropocentric Knowledge Discovery in Database) que nous avons développée. Elle est dérivée de la méthode KDD à laquelle nous avons ajouté l'aspect incrémental et l'aspect centré sur l'utilisateur. Nous présentons, en particulier, une technique fondée sur le principe du tri par paquets permettant d'extraire efficacement les symboles graphiques contenus dans un document PDF. Elle est comparée aux résultats de Puglisi sur les chaînes de caractères. Puis, nous formulons l'hypothèse selon laquelle la prise en compte de l'ordre chronologique présent dans les fichiers PDF et dans le processus incrémental, améliore la restructuration des documents. Nous montrons la validité de cette hypothèse sur un certain nombre d'exemples. Enfin, nous montrons l'efficacité du processus pour identifier les symboles en même temps que les équipotentielles. Nous terminons le mémoire en montrant les avancées et les limites de la solution de la méthode (A)KDD et nous proposons des perspectives.

Les principales contributions sont :

L'hypothèse de l'ordre chronologique prise, pour la première fois, comme principe de base pour restructurer des documents déstructurés.

La transformation d'un espace 2D, qui est habituellement l'espace dans lequel sont décrits les schémas, les plans ou les dessins, en un espace 1D qui est matérialisé par une chaîne de codes. Cette transformation 2D en 1D n'est possible et n'a un intérêt que grâce à l'hypothèse décrite ci-dessus.

L'adaptation de la méthode KDD à la fouille de données graphiques, alors que cette méthode était initialement orientée base de données, afin de trouver les règles d'apparition de répétitions des données dans ces bases (détections de motifs à l'intérieur d'une chaîne, nettoyage des données d'entrée).

L'ajout de l'aspect incrémental à la méthode KDD et de l'aspect anthropocentré (méthode (A)KDD) pour la fouille des données graphiques. L'aspect incrémental permet de diminuer le volume des données à traiter (c'est une préoccupation permanente de ces travaux) en remplaçant N données graphiques par un élément structurant de plus haut niveau. L'aspect centré utilisateur permet d'avoir un contrôle permanent de l'utilisateur et ainsi de s'assurer que la restructuration est conforme à ce que souhaite l'utilisateur.

Abstract

This thesis deals with the restructuring of unstructured PDF documents containing graphical elements such as schematics, plans and drawings, with the aim of restructuring them. Using the KDD (Knowledge Discovery in Database) method for data restructuring, we introduce the (A)KDD (Antropocentric Knowledge Discovery in Database) method that we developed which is derived from the KDD method by adding an incremental aspect and an user-centered approach. We present, in particular, a technique based on on the bucket sort algorithm pattern in order to extract with efficiency graphic symbols contained in a PDF file. It is compared to the results obtained by Puglisi on strings. Then, we formulate the hypothesis : " taking into account the chronological order present in the PDF files in the incremental process improves the restructuring of the documents ". We illustrate the validity of this hypothesis on several examples. Finally, we show the efficiency of the process in the identification of the symbols at the same time as the equipotentials. The thesis concludes by showing the advances and the limits of the solution of the (A)KDD method and we propose some perspectives.

The main contributions are :

This is the first time that the chronological order hypothesis is taken as a basic principle for restructuring unstructured documents.

The transformation of a 2D space, which is usually the space in which the diagrams, plans or drawings are described, into a 1D space that is materialized by a code string. This 2D transformation in 1D is possible and is only interesting thanks to the hypothesis described above.The adaptation of the KDD method to the search of graphical data whereas this method was initially database oriented in order to find the rules of data repetitions detection in these bases (detections of patterns inside a string, cleaning input data).

The addition of the incremental aspect to the KDD method and the anthropocentric aspect (method (A)KDD) for the extraction of graphic data. The incremental aspect makes it possible to reduce the volume of the data to be processed (it is a permanent concern of this work) by replacing n graphical data by one structuring element of higher level. The human centered aspect allows a permanent control of the user and thus ensure that the restructuring is as required by the user.

Préambule

Pourquoi à 65 ans faire une thèse sur la restructuration des documents déstructurés ?

Les raisons sont multiples, je vais les développer dans ce préambule.

35 ans de déstructuration des documents graphiques

En 35 ans, nous avons, avec mon épouse, créé et géré trois sociétés dans le domaine du Dessin Assisté par Ordinateur (DAO) et de la Conception Assistée par Ordinateur (CAO).

Notre première société, créée en 1984, en région parisienne, s'appelait Intelligence Artificielle en Micro-Informatique (IA-MICRO™). Aujourd'hui on entend et on voit partout ces deux mots, Intelligence Artificielle, mais en 1984, ils n'avaient pas le même impact marketing.

Notre deuxième société, Algo'Tech Informatique™, a été créée en 1999, ici, au Pays Basque, dans l'incubateur de l'ESTIA, du fait de la présence de l'ESTIA et du laboratoire.

Notre troisième société 1A3i™ (Applications & Innovations Industrielles & Informatiques) a été créée en Mai 2017, elle aussi dans l'incubateur de l'ESTIA. 1A3i exploitera les résultats de la recherche faîte dans le cadre de cette thèse, après industrialisation.

Dans nos deux premières sociétés nous avons développé des logiciels de CAO et de DAO. Plus particulièrement des logiciels de CAO pour la réalisation des schémas électriques, des systèmes d'automatisme et d'électromécanique. Ces logiciels servaient également à la conception des schémas hydrauliques, pneumatiques, climatiques, électroniques, des organigrammes, des logigrammes et ils permettaient la réalisation des plans des bâtiments ou des armoires électriques pour implanter les équipements.

Pendant 35 ans, nous avons structuré, dans nos logiciels, toutes les informations graphiques et textuelles qu'utilisaient nos clients au travers de nos logiciels..

Pendant 35 ans, nous avons également demandé à nos équipes de développeurs de déstructurer ces données structurées. On peut se poser la question : pourquoi déstructurer des informations qui sont structurées ? Les réponses sont dans l'annexe A que l'on peut résumer en la nécessité d'obtenir des informations graphiques basiques compréhensibles par les périphériques d'affichage et d'impression (cartes graphiques, traceurs, imprimantes) et par la volonté de nos clients de disposer d'un support ayant les mêmes possibilités que le papier (voir l'information sans pouvoir l'exploiter « facilement »).

Ayant déstructuré cette information structurée, je sais pourquoi on la de-

structure, je sais comment on la destructure, je sais comment on la stocke. Cela me donne de très bonnes bases pour savoir comment restructurer cette information et ce que l'on doit obtenir après restructuration. Il suffit de faire le chemin inverse de la déstructuration (facile à dire, mais plus difficile à faire).

Une conviction profonde sur la nécessité de faire de la recherche

Les points ci-dessous de mon parcours professionnel m'ont amené à considérer que la recherche est indispensable. En effet, nous avons, avec mon épouse :

- Développé pendant plus de 35 ans des sociétés de conception de logiciels,
- Participé à une dizaine de projets de recherche avec :
  - Les plus grandes sociétés (Airbus, Safran, Turbomeca, Bull, ...),
  - Les plus grands laboratoires (l'Onera, le CEA, l'Inria de Bordeaux, le LORIA de Nancy, le LaBRI de Bordeaux, l'ESTIA, L'Enit de Tarbes,...),
- Initialisé 7 thèses CIFRE :
  - 2 dans le domaine de la reconnaissance des schémas électriques sur support papier(voir l'article [GT05] et la thèse [Gab08]),
  - 2 dans le domaine du génie logiciel,
  - 2 dans le domaine de la simulation électrique (voir les articles [Leg+04], [Leg+03] et la thèse [Leg04]) et de la simulation électromagnétique (voir l'article [COV17]),
  - 1 sur les réseaux de neurones (voir l'article [KRA17] et la thèse [Kaa18]).

Je suis convaincu qu'une entreprise, et plus particulièrement les Petites et Moyennes Entreprises PME, ou les start-ups, dans le domaine des nouvelles technologies, doivent être à la pointe de l'innovation. Pour être à la pointe de l'innovation, il faut faire de la recherche. Cette recherche est très difficile à mettre en place dans les PME, le quotidien prime sur la vision à moyen ou long terme. C'est pour cette raison que j'ai commencé la thèse avant de développer commercialement la société 1A3i™ . Dans l'annexe B, j'explique plus en détail pourquoi, à 65 ans, j'ai fait ce choix.

L'absence de recherche sur les documents déstructurés graphiques

La recherche scientifique est inexistante au niveau de la restructuration des fichiers vectoriels, contenant des dessins, des schémas. À partir des années 2000 (et même un peu avant), de nombreuses équipes ont travaillé sur la restructuration des fichiers vectoriels PDF contenant du texte, des tableaux, des articles de presse, des articles scientifiques. Ces travaux de recherche ont permis, à des éditeurs de logiciels, de proposer des solutions pour transformer les fichiers vectoriels PDF, qui contiennent du texte ou des tableaux, en fichier Word™ ou Excel™.

À notre connaissance, les seuls travaux existants sur la restructuration des documents graphiques, en particulier des schémas électriques, sont les travaux menés par l'équipe Mnémosime de l'Inria de Bordeaux. Ces travaux de recherche [KRA17] et la thèse d'Ikram Chraibi Kaadoud [Kaa18] abordent le problème de la structuration d'un dossier électrique. En suivant les renvois des équipotentielles et les informations textuelles présentes dans les fichiers PDF, ils restructurent le dossier électrique au niveau des grandes fonctions de ce dossier.

C'est la septième et dernière thèse CIFRE que j'ai initialisée au sein de la société Algo'Tech Informatique™ et qui a été soutenue en 2018. Cette thèse aborde le problème de la restructuration du dossier électrique par le texte et non pas par les objets graphiques. A ma connaissance, il n'existe aucun travail scientifique sur la restructuration des fichiers PDF (ou d'autres types de format de fichier vectoriel) qui contiennent des schémas électriques.

Le volume des données déstructurées

Les chiffres suivants sont très significatifs :

En 2020, 99% de l'information sera numérique [GR12]. Elle est produite par des logiciels présents sur les ordinateurs, les téléphones portables, les tablettes, les voitures, etc. Soit presque tous les équipements.

En 2020, 75% de cette information seront stockés sous forme non structurée. Ces 75% seront difficilement exploitables, et même inexploitables. On peut consulter, visualiser, imprimer, cette information, qu'elle soit graphique ou textuelle, mais elle est inutilisable, sauf si elle est ressaisie dans un nouveau logiciel ou si des outils permettent de la restructurer.

En 2020, le volume des données numériques sera de 40 zettaoctets (zettaoctet (Zo) = 10^{21} octets, = 1 000 000 000 000 000 000 000 octets) voir l'étude de 2012 [GR12] et celle de 2014 [Tur+14]. Ce volume, ramené au nombre d'habitants, représentera environ 7 à 8 téraoctets (téraoctet (To) = 10^{12} octets = 1 000 000 000 000 octets) par habitant de la planète en 2020.

A partir de 2025, le volume des données numériques sera multiplié par 2 tous les 9 mois. Cela veut dire que tous les 90 mois (soit 7ans et 6 mois) le volume sera multiplié par 1024 (2^{10}) et que, 15 ans après (soit en 2040), le volume des données existant en 2025 sera multiplié par plus de 1 000 000 de fois (2^{20}). Ce coefficient multiplicatif de 1 million est tellement grand qu'on peut imaginer que les données seront stockées sous une autre forme.

Ces volumes sont gigantesques. Microsoft™ et Adobe™ ne s'y sont pas trompés en proposant de restructurer les informations textuelles (pour Word™) ou les tableaux (pour Excel™) contenus dans les fichiers PDF.

L'objectif des travaux de cette thèse est de mettre en place les premières méthodologies pour restructurer les documents graphiques déstructurés vectoriels, comme cela a été fait dans les années 2000 à 2010 pour les textes et les tableaux, puis après industrialisation, permettre la réutilisation de ces fichiers graphiques dans des logiciels de DAO, CAO et PAO.

Développer l'emploi au Pays Basque

L'objectif final de ces travaux de recherche est de pouvoir développer, ici, au Pays Basque, l'emploi, près de l'école et du laboratoire de l'ESTIA. C'est ce que nous savons faire avec mon épouse. En tant qu'ancien « industriel » je ne peux pas concevoir la recherche sans son application concrète, avec pour objectif de créer de l'emploi.

La thèse sera suivie de deux ans d'industrialisation des travaux initialisés pendant cette thèse. La thèse développe une méthodologie et lève certains verrous technologiques pour arriver à restructurer les documents graphiques. Les travaux de recherche faits pour restructurer les documents PDF qui contenaient des données textuelles ou des tableaux n'avaient que pour seul objectif de lever les verrous technologiques et de proposer des méthodes pour restructurer ces données. Ce n'est qu'après ces travaux, et sûrement en les utilisant, que Microsoft et Adobe ont développé des applicatifs qui permettent d'importer ces fichiers dans les logiciels de bureautique.

Les deux ans d'industrialisation serviront à faire des preuves de concept sur la restructuration des documents déstructurés graphiques, sur des cas concrets. Ces preuves de concepts aboutiront à des produits qui seront commercialisés par la société 1A3i. C'est 1A3i qui finance l'ensemble de ces travaux de recherche.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

(A)KDD	*Antropocentric Knowledge Discovery in Database*
3DS	3D Studio
ADA	Langage, dont le nom correspond au prénom d'Ada Lovelace
ADN	Acide DésoxyriboNucléique
ASCII	*American Standard Code for Information Interchange*
BMP	*BitMaP*
CAO	Conception Assistée par Ordinateur
CEI	Commission Électrotechnique Internationale
CES	*Consumer Electronics Show*
CGM	*Computer Graphics Metafile*
CIDRE	Coopérative Interactive Document Reverse Engineering
CIFRE	Convention Industrielle de Formation par la REcherche
COG	*Component Object Graphics*
COMAU	*COnsorzio MAchine Utensili*
CPU	*Central Processing Unit*
DAO	Dessin Assisté par Ordinateur
DBSCAN	*Density-Based Spatial Clustering of Applications with Noise*
DM	*Data Minning*
DOLORES	*DOcument LOgical REStructuring*
DXF	*Drawing eXchange Format*
DPI	*Dots Per Inch*
DVD	*Digital Versatile Disc*
DW	*Data Warehouse*
DWF	*Design Web Format*
DWG	*DraWinG*
DWT	*DraWing Template*
ECD	Extractions des Connaissances des Données
ECML-PKDD	*European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*
EGC	Extractions et Gestion des Connaissances
eGSA	*external memory Generalized Suffix Array*
EIS	*Executive Information Systems*
EMF	*Enhanced MetaFile*
ERP	*Enterprise Requirement Planning*
ETI	Entreprises de Taille Intermédiaire

EPDF	Encapsulated **PDF**
FIAT	*Fabbrica Italiana Automobili Torino*
GAFA	Google™, Apple™, Facebook™ et Amazon™
Go	Giga octets = 1 000 000 000 octets
GPU	*Graphics Processing Unit*
HPGL	*Hewlett Packard Graphic Language*
IADM	*Incremental Adaptive Deep Model*
IBM	*International Business Machine*
ICAI	*International Conference on Applied Informatics*
ICDAR	*International Conference on Document Analysis and Recognition*
IDC	*International Data Corporation*
IGES	*Initial Graphics Exchange Specification*
IHM	Interface Homme Machine
ISO	*International Organization for Standardization*
JPEG	*Joint Photographic Experts Group*
KDD	*Knowledge Discovery in Database*
LE	*Left Extendible*
LCP	*Longest Common Prefix array*
OCR	*Optical Caracter Recognition*
OLAP	*On-Line Analytic Processing*
NAIA	Forum Néo-Aquitain de l'Intelligence Artificielle
NE	*No Extendible*
NLE	*No Left Extendible*
NRE	*No Right Extendible*
PAO	Publication Assistée par Ordinateur
PC	*Personal Computer*
PCL	*Printer Command Language*
PCRD	Les Programmes-Cadre pour la Recherche et le Développement technologique
PDF	*Portable Document File*
PID	*Process and Instrumentation Diagram*
PME	Petites et Moyennes Entreprises
PROLOG	PROgrammation LOGique
PS	*PostScript*
R2E	Réalisations et Études Électroniques
RAM	*Random Access Memory*
RE	*Right Extendible*
SA	*Suffixe Array*
SAP	*Suffixe Array for Pattern*
SATT	*Sociétés d'Accélération du Transfert de Technologies*

SIGKDD	*Special Interest Group on Knowledge Discovery and Data Mining*
SLT	*StereoLiThography file format (ANSI and binary)*
SOFINS	Spécial Opérations Force Innovation Network Séminar
STEP	*Standard for The Exchange of Product model data*
SVG et SVGZ	*Scalable Vector Graphics*
TIFF	*Tagged Image File Format*
TTT	*Type by Task Taxonomy of information visualization*
UTF-8	*Universal Character Set Transformation Format - 8 bits*
VRML	*Virtual Reality Modeling Language*
WMF	*Windows MetaFile*
XCDF	*Canonical and Structured Document Format*

Section 1

Introduction

Depuis des millions d'années, l'homme produit de l'information. Aujourd'hui cette information est de plus en plus numérique et croît de façon exponentielle. Un des problèmes liés à cette information numérique et à sa croissance est que plus de 75% des informations sont inexploitables par les ordinateurs. C'est un réel problème vu le volume des données. Face à ce défi, de nombreuses équipes de recherche se sont intéressées, depuis les années 2000, à la restructuration de cette information non structurée. Mais, à notre connaissance aucune thèse ne s'est intéressée à la restructuration des documents déstructurés de type PDF contenant des éléments graphiques tels que les schémas, les plans et les dessins, dans l'objectif de les restructurer.

1.1 La notion de document

La définition, de la notion de document, donnée par Michael K.Buckland [Buc97] en français, dans « *What is a document ?* », est la suivante : « Document : Toute base de connaissance, fixée matériellement, susceptible d'être utilisée pour consultation, étude ou preuve. Exemples : manuscrits, imprimés, représentations graphiques ou figurées, objets de collection, etc ... ».

Un document, c'est donc une information sur un support. Cette information est persistante, et a une valeur explicative, descriptive ou de preuve. Cette définition est très large. Si on la place dans l'évolution de l'humanité (voir annexe C) on constate que le développement de cette notion de document (information sur un support) ne fait qu'évoluer et croître de façon très importante en volume. D'abord en passant des silex aux peintures rupestres, des peintures rupestres au parchemin, du parchemin au papier et récemment du papier au support numérique. Sous cette forme numérique, le volume des documents générés ne fait que progresser et cette progression est exponentielle.

1.2 La structuration des documents

Dans le préambule, nous avons souligné le fait qu'en 2020, les documents numériques représenteront 99% de la totalité des documents existant dans le monde et que 75% ne sont pas structurés. Ce sont ces 75% de documents qui nous intéressent dans le cadre de cette thèse. Les documents numériques sont classés en trois grandes familles, les documents structurés, les documents non structurés et les documents semi-structurés (voir « *EASE : An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data* » [Li+08]), auxquelles nous ajoutons la famille des documents déstructurés.

Ces notions de documents structurés, non structurés et semi-structurés sont maintenant très utilisées dans le domaine informatique et, également, dans le milieu industriel. Nous allons les expliciter.

Les trois notions, documents, informations, données, représentent la même chose avec un niveau de granularité différent. Un document est une information qui contient des données. Si le document est structuré, l'information et les données sont structurées. Réciproquement, si les données sont structurées, l'information et le document sont structurés.

1.2.1 Le document structuré

Pour le document structuré nous prenons la définition de Baarset et Kemper dans « *Management support with struc-tured and unstructured data—an integrated business intelligenceframework* ». La définition est la suivante : « *Structured data is here understood to be data that is assigned to dedicated fields and that can thereby be directly processed with computing equipment* » [BK08]. Donc, Un document structuré est un document facilement exploitable de façon informatique avec des matériels informatiques. Par exemple, une base de données est un document structuré. Un programme peut assez facilement exploiter l'information structurée contenue dans la base de données.

Negash et Gray dans « *Business intelligence* » [NG08] donnent comme exemple de données structurées « *OLAP = On-Line Analytic Processing, DW=Data Warehouse, DM=Data Mining, EIS = Executive Information Systems, and ERP = Enterprise Requirement Planning* ».

1.2.2 Le document non structuré

Un document numérique non structuré est un document qui n'est pas exploitable, ou très difficilement exploitable, par un ordinateur. Un document numérique non structuré est un document qui peut être affiché, imprimé ou échangé. Toutes les personnes ayant des connaissances du domaine peuvent « comprendre » et exploiter le contenu de ce document une fois imprimé ou affiché. De façon interne, ce document ne contient aucune donnée structurée

exploitable directement par la machine. Par contre, l'homme de l'art peut identifier tous les éléments, voir et comprendre la structure de l'information.

Dans « *Predictive business process monitoring with structured and unstructured data* » [Tei+16] les auteurs donnent le texte libre comme exemple de données sans structure. Ils considèrent même qu'un champ texte d'une base de données, pour enregistrer par exemple un commentaire libre, contient des données déstructurées et que, par conséquent, une telle base de données peut être considérée comme un document déstructuré si l'on veut exploiter ce champ commentaire.

1.2.3 Le document semi-structuré

Si l'on prend un fichier Word™, c'est un fichier qui peut être structuré dans sa forme. On va trouver un titre, des sous-titres, des paragraphes, la table des matières, des figures, des images, des légendes, des annotations, donc il a une structure. Mais, il est déstructuré dans son contenu. Les paragraphes sont des textes libres qui sont sans structure. Ce document peut être lu par un programme informatique, le programme peut retrouver facilement la structure (titres, sous-titres, paragraphes, etc...) mais les informations des paragraphes sont « verrouillées » car sans structure. Il est donc structuré pour une partie de son information et déstructuré pour une autre partie, on dit qu'il est semi-structuré.

Dans « *FlashRelate extracting relational data from semi-structured spreadsheets using examples* », les auteurs ([Bar+15]) prennent comme exemple de données semi-structurées les feuilles de calcul du logiciel Excel. Ils considèrent qu'un logiciel va pouvoir lire ces tableaux, trouver les lignes et les colonnes, mais que les données sont « *Locked-in* » (verrouillées).

1.2.4 Le document déstructuré

Un document déstructuré et un document qui, initialement, était structuré ou semi-structuré, et que l'on a transformé en document non structuré.

C'est, par exemple, le cas de tous les schémas ou les plans produits par les logiciels de Dessin Assisté par Ordinateur (DAO), de Publication assistée par Ordinateur (PAO), et encore plus par les logiciels de Conception Assistée par Ordinateur(CAO). Ces logiciels conservent, dans leur fichier interne, l'ensemble de l'information et des données sous une forme très structurée, un peu comme une base de données. D'ailleurs beaucoup de logiciels de DAO/CAO/PAO utilisent des bases de données pour conserver leurs données internes.

Mais, comme nous l'avons décrit, dans le préambule, quand on veut imprimer, conserver ou transmettre ces documents, les éditeurs des logiciels déstructurent les données structurées qu'ils ont créées avec leur logiciel. Ainsi ils obtiennent les éléments graphiques les plus basiques et les plus simples possibles. Ces éléments sont compréhensibles par le périphérique de sortie qui visualise l'information (table traçante, imprimante, écran, etc.) et ils permettent

à l'homme de voir et de « comprendre » l'information contenue dans le document. L'exploitation de ces éléments basiques est impossible par des logiciels, sauf si on restructure l'information. Et c'est l'objectif de nos travaux.

Un autre exemple de documents déstructurés : tous les documents générés par la bureautique et stockés au format PDF comme, par exemple, les documents Word™, Excel™, ou PowerPoint™. Ces documents, quand ils sont aux formats de Word™, Excel™ ou PowerPoint™ sont des documents semi-structurés accessibles par des ActiveX depuis un programme informatique.

1.2.5 Mégadonnées (*big data* et *small data*)

Le terme anglais « *big data* », ou, en français, « mégadonnées » ou encore « données massives », est associé à l'ensemble de ces données qui ne font que croître, et que l'homme, et même la machine, ont des difficultés à exploiter. Ces « données massives » ne sont l'apanage que de quelques très grandes entreprises, en général les GAFA (Google™, Apple™, Facebook™et Amazon™) ainsi que des plus grandes entreprises mondiales. Les autres entreprises ont beaucoup plus de mal à disposer de données que l'on qualifierait de « données massives ». C'est pour cette raison que le terme anglais « *small data* » est apparu, la traduction en français est « petite quantité de données » ou « données non massives ».

Ces 2 notions « *big data* » et « *small data* » sont très bien expliquées dans « *Small data in the era of big data* » [KL15]. Le tableau synthétique ci-dessous est repris des travaux de Kitchin et Lauriault, il montre bien les différences entre ces 2 notions :

Caractéristiques	*small data*	*big data*
Volume	Limité	Très grand
Exhaustivité	Échantillons	Population entière
Résolution	Grossière	Très fines
Indexicalité	Faible	Forte
Relationnalité	Faible	Forte
Rapidité (*Velocity*)	Lente	Forte
Diversité	Limitée	Large
Adaptabilité	Basse à moyenne	Haute
Passage à l'échelle	Basse à moyenne	Haute

TABLE 1.1 – Comparaison entre « *big data* » et « *small data* »

Le critère principal qui différentie le « *big data* » du « *small data* », est le volume d'informations disponibles. Les PME et les ETI, ne disposent pas d'un très grand nombre de données. Si l'on prend, par exemple, le domaine de la fabrication de machines, une ETI spécialisée dans ce domaine, va créer entre 10 et 50 dossiers différents par an, ce qui représentera, après 50 ans d'activité,

entre 500 et 2 500 dossiers. Nous sommes très loin des dizaines ou centaines de millions de données que manipulent le « *big data* ». Nous verrons que les solutions développées pour le « *big data* » ne peuvent pas s'appliquer à notre problématique.

L'un des objectifs de ce travail de recherche est de développer, après industrialisation, des outils de restructuration de l'information. Les cibles que nous visons, en priorité, sont les PME et les ETI. Dans ces tailles d'entreprises, quand on parle des données disponibles pour un problème, on parle plus de « *small data* » que de « *big data* ».

1.3 La restructuration des documents déstructurés : Pourquoi ? Quels documents ? Comment ?

Maintenant nous allons répondre aux trois questions fondamentales suivantes :

— Pourquoi ? : Pourquoi restructurer des documents déstructurés ?
— Quels documents ? : Quels types de documents sont concernés par nos travaux ?
— Comment ? : Comment restructurer les documents déstructurés ?

1.3.1 Pourquoi restructurer des documents déstructurés ?

Il y a une richesse inestimable cachée à l'intérieur de ces documents déstructurés. Aujourd'hui, l'exploitation de ces documents se fait manuellement, ce qui prend beaucoup de temps et peut être source d'erreurs. Vu l'augmentation constante du volume de ces données, leur exploitation ne sera plus possible manuellement. Il faudra l'aide de la machine pour pouvoir analyser le contenu de ces fichiers déstructurés, afin d'exploiter leur contenu.

Le changement de support (du papier au support numérique) apporte de très grandes opportunités pour l'exploitation de l'information contenue dans ces documents. L'exploitation automatique ou semi-automatique des documents sur support « papier », par l'ordinateur, a été un sujet de recherche très important pendant ces 50 dernières années. Les résultats obtenus dépendaient énormément de la qualité initiale du document et de sa digitalisation pour son exploitation par informatique.

Après 50 ans de recherche, les logiciels répondant à cette problématique sont quasiment inexistants, sauf dans le domaine de la reconnaissance des textes imprimés ou manuscrits.

Le fait que le support initial soit sur un support informatique (issu d'un logiciel) change beaucoup de choses par rapport au support papier. Ce changement de support ouvre un nouveau champ de recherche aussi important que celui consacré à la transformation du support papier en support numérique «

intelligent », d'autant plus que le support informatique a été produit par une machine, selon un schéma bien précis et de façon répétitive.

La méthodologie que nous allons développer va permettre de restructurer l'information exactement comme le fait un expert du domaine. Un exemple de cette restructuration par un expert est donné dans l'annexe D.

Les applications de la restructuration sont principalement : la rétro-ingénierie de documents (*reverse engineering*), la comparaison de plusieurs documents, la classification et l'organisation de tous les documents, l'indexation et la recherche à l'intérieur de documents.

1.3.1.1 La rétro-ingénierie de documents (*reverse engineering*)

La rétro-ingénierie pour l'analyse de documents consiste à réintroduire manuellement ou automatiquement, les notions contenues dans ces documents dans un nouvel environnement (en général un logiciel). La restructuration que nous proposons a pour objectif de les réintroduire automatiquement.

Les raisons de la rétro-ingénierie sont diverses, soit le logiciel n'existe plus, soit le logiciel ne peut pas relire d'anciennes versions, soit c'est un document produit par quelqu'un d'autre. On pense que la rétro-ingénierie est un procédé interdit. C'est faux. Beaucoup de pays autorisent, par la loi, la rétro-ingénierie pour des raisons d'interopérabilité. C'est le cas de la France, ce droit de rétro-ingénierie est inscrit dans l'article L122-6-1 du code de la propriété intellectuelle. Ce droit est également inscrit dans les directives du Parlement Européen. Malgré ce droit, de nombreux éditeurs de logiciels n'hésitent pas à inclure des clauses interdisant la rétro-ingénierie, dans les conditions d'utilisation de leurs logiciels, mais ces clauses sont illicites (en France et en Europe) donc caduques.

1.3.1.2 La comparaison de plusieurs documents

Il s'agit de pouvoir comparer automatiquement plusieurs versions de fichiers différents, correspondant à des évolutions ou des modifications du document au cours du temps. Ceci est possible après restructuration de tout ou partie de l'information. La comparaison peut se faire à plusieurs niveaux, suivant le degré de reconstruction de l'information.

C'est une application que nous avons développée dans le cadre de cette thèse pour la société COMAU™ (*COnsorzio MAcchine Utensili* : 10 000 personnes dans le monde) qui appartient au groupe FIAT™ (*Fabbrica Italiana Automobili Torino* : 230 000 personnes dans le monde). En effet, les besoins du bureau d'études de COMAU™ étaient les suivants : comparer la version « x » avec la version « y » d'un même document, identifier rapidement les différences, recenser les différences dans un rapport.

1.3.1.3 La classification et l'organisation de tous les documents

Le volume grandissant des documents de tout type va nécessiter des outils de classification et d'organisation des documents automatique ou semi-automatique. Il va bientôt être très difficile de classer manuellement la totalité des documents déstructurés qui seront en notre possession. La restructuration permet d'automatiser cette classification après apprentissage d'un certain nombre de documents. Les règles de classification se déduisent automatiquement de cet apprentissage.

1.3.1.4 L'indexation et la recherche dans plusieurs documents

La recherche de l'information dans les documents est un point fondamental. Le rapport IDC™ (International Data Corporation) indique que 75% de l'information en notre possession est déstructurée. De plus, cette information est très souvent dupliquée : une étude récente du groupe IDC™, pour Seagate™ [Buc97], montre que la même information pouvait être présente, au sein d'une entreprise, entre 10 et 100 fois sur les différents ordinateurs du personnel de la société (mails, téléchargements, archivage, duplication, etc). Et ce même fichier peut exister sous 2 à 20 versions différentes.

Un outil qui indexe la totalité des informations déstructurées sera une nécessité pour les entreprises.

1.3.1.5 Les volumes

Les volumes que nous avons indiqués dans le préambule sont gigantesques, et basés sur des hypothèses établies en 2012 et 2014.

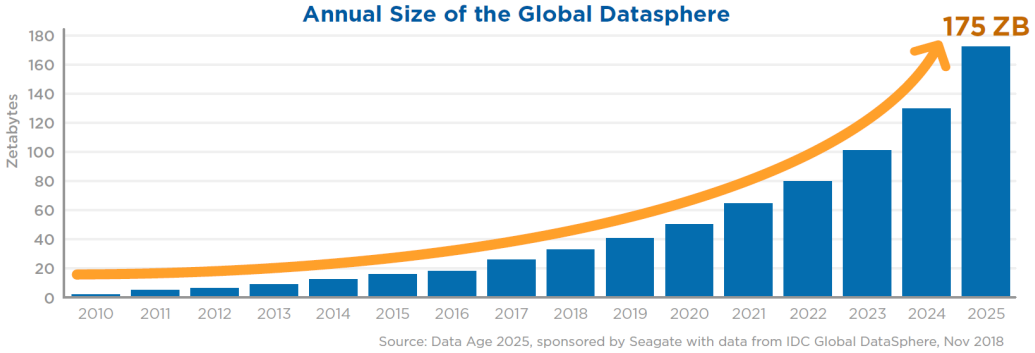


FIGURE 1.1 – Évolution annuelle du volume des données : source IDC Novembre 2018.

Ces chiffres (voir figure 1.1) sont confirmés par la dernière étude d'IDC™ de Novembre 2018 qui confirme le chiffre de 40 zettaoctets pour 2020, et prévoit 175 zettaoctets pour 2025 [Tur+14]. Ce chiffre de 175 zettaoctets est tellement grand que les auteurs du rapport donnent quelques exemples de ce qu'il peut représenter. Si ces données étaient enregistrées sur des DVD (Digital Versatile

Disc ou disque numérique polyvalent) et que l'on empile ces DVD, cela représenterait 23 fois la distance terre-lune ou 222 fois le tour de la terre. Si une personne téléchargeait l'ensemble de ces données avec un débit de 25 Megabits/seconde il lui faudrait 1,8 milliard d'années.

1.3.2 Quels sont les types de documents concernés par nos travaux ?

Ce qui nous intéresse, dans le cadre de cette thèse et de ces travaux de recherche, c'est de restructurer les documents graphiques déstructurés de type vectoriel. Le type de fichiers que nous allons traiter dans cette thèse sont les fichiers PDF qui contiennent du texte et des dessins. Dans l'annexe D.3 nous donnons une liste des autres types de fichiers vectoriels que nous traitons.

Le plus simple est de prendre un exemple pour visualiser le type de documents que nous restructurerons et de définir ce que doit apporter cette restructuration.

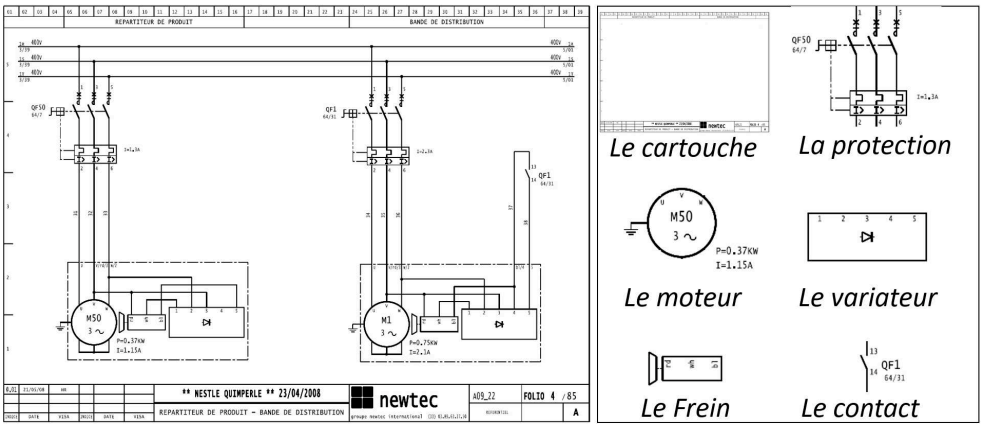


FIGURE 1.2 – Exemple schéma électrique, à gauche le folio, à droite les symboles

Dans la figure 1.2 nous avons, sur la partie gauche, la vue d'un folio d'un schéma électrique, et sur la partie droite les principaux symboles du folio. Dans le cadre de ces travaux de recherche, nous développons la possibilité d'identifier, entre autres, tous ces symboles qui sont présents sur ce folio et, de façon plus générale, dans le dossier.

L'objectif de cette thèse est de définir la méthodologie pour restructurer les documents déstructurés de type vectoriel contenant des informations graphiques et textuelles.

Dans cet exemple, on voit que les parties graphique et textuelle sont présentes dans tous les documents. Les textes (parfois des chiffres ou des abréviations) sont peu nombreux, mais présents. Ils sont en nombre plus ou moins important suivant le type de schémas. Par contre, on constate que le texte et les dessins sont intimement liés. On n'a pas, d'un côté des textes, et d'un autre côté,

dans des zones bien spécifiques, des dessins. Les textes et les graphismes sont mélangés. Les textes sont associés à des graphismes pour former des symboles.

Voici un autre exemple (voir figure 1.3, page 9) il est extrait d'un article scientifique où l'on peut identifier des résistances, des capacités et des diodes.

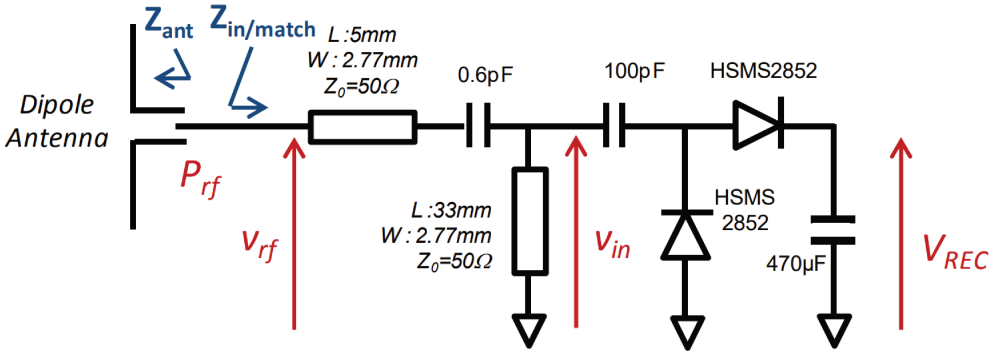


FIGURE 1.3 – Schéma extrait de « *A 900MHz RF Energy Harvesting Module* » Thierry Taris, Valérie Vigneras, Ludivine Fadel [TVF12]

1.3.3 Comment restructurer les documents déstructurés ?

La restructuration des documents déstructurés est très complexe, car elle est composée d'une succession d'opérations qui ne peut être faite qu'avec l'aide de l'utilisateur. Elle n'est pas automatisable si l'on souhaite conserver un taux de réussite de la restructuration supérieur à 95% et si possible proche des 100%.

1.3.3.1 L'aspect incrémental de la restructuration des documents déstructurés

Restructurer un document déstructuré, c'est comme un jeu de LEGO™. On a toutes les pièces en vrac et il faut, à partir de ces pièces en vrac, reconstituer l'information. C'est un processus incrémental qui doit se faire étape par étape et qui se décompose en cinq grandes étapes pour les schémas [1] : la restructuration des éléments graphiques vectoriels de base, la restructuration des éléments textuels, la restructuration des symboles et des liaisons (équipotentielles, tuyaux, murs etc...), la restructuration des fonctions, et pour terminer la restructuration du dossier. Ces cinq grandes étapes peuvent changer légèrement d'un type de schéma (document graphique) à l'autre. L'étape la plus stratégique est la restructuration des symboles et des liaisons, elle ne peut être réalisée qui si les étapes précédentes ont été faites. Pour la clarté de la thèse,

1. Schémas électriques, hydrauliques, pneumatiques, électroniques, d'automatisme, d'électromécanique, d'électronique, de tuyauterie, d'instrumentation (PID : *Process and Instrumentation Diagram*), bâtiments, etc...

nous prendrons, dans la mesure du possible, des exemples dans le cadre des schémas d'automatisme et d'électromécanique.

1.3.3.2 L'importance de l'homme dans la restructuration.

Prenons l'exemple de l'outil [2] Scan'Builder™ [3] développé pour introduire automatiquement les plans papiers dans un logiciel de DAO/CAO (logiciel de la gamme des solutions d'Algo'Tech Informatique™). C'est cette notion « d'introduction automatique » qui a eu des conséquences très importantes sur l'acceptabilité du logiciel par les utilisateurs. L'outil était conçu, au départ, pour que l'utilisateur n'intervienne pas, ou le moins possible, dans le processus de restructuration de l'information. Ceci a été une très grande erreur, et je vais expliquer pourquoi.

Pour restructurer les schémas, présents sur un support papier, nous avions une quinzaine d'opérations importantes qui s'enchaînaient les unes après les autres. Ces 15 opérations avaient des taux de reconnaissance qui étaient compris entre 90% et 99%, selon le cas. C'était un très beau résultat quand on le comparait aux meilleurs résultats scientifiques de chaque opération à l'époque. Mais, quand on effectue l'ensemble de ces opérations, les unes après les autres, les taux d'erreurs se cumulent et peuvent même s'amplifier, ce qui fait que, dans certains cas, suivant la qualité des documents en entrée, on avait un taux d'erreurs cumulées qui pouvait approcher les 30%, et parfois même le dépasser. Sur des documents de très bonne qualité, nous avions des taux de reconnaissance de l'ordre de 95% ce qui était excellent. Mais malheureusement nos clients n'avaient pas toujours des documents d'excellente qualité.

Ces 30% de taux d'erreurs nous ont obligés à revoir en profondeur l'ensemble du logiciel. En effet, modifier un élément sur 3, pour le corriger, était inacceptable, l'utilisateur passait trop de temps à faire ces corrections, ce qui pouvait prendre plus de temps que s'il avait dessiné directement le schéma. Les modifications que nous avons effectuées ont consisté à faire intervenir l'utilisateur au plus tôt dans la correction des erreurs. En effet, plus l'erreur est corrigée tôt, plus elle est simple à corriger, et moins elle a de répercussion dans la suite du traitement puisqu'on l'a supprimée. Mais cela a été très difficile, car il a fallu repenser complètement la méthode de restructuration des documents, et réécrire des parties entières du logiciel. Ce qui s'est révélé parfois impossible pour des raisons de coûts.

C'est une erreur que nous n'avons pas voulu reproduire pour restructurer les documents déstructurés de type vectoriel. Les étapes et les opérations sont presque les mêmes, la différence vient de l'information de base que l'on traite. Pour Scan'Builder™ c'étaient des images matricielles, dans le cadre de cette thèse ce sont des vecteurs. Dès le départ, nous avons conçu la solution en met-

2. Scan'Builder™ permet de lire les fichiers images (TIFF, BMP, JPEG,...) qui contiennent des schémas électriques et de les introduire dans un logiciel de DAO comme si un utilisateur avait fait le dessin.

3. Ce développement a fait l'objet de deux thèses CIFRE et d'un programme de recherche financé par l'Europe dans le cadre du 7e PCRD (Programmes Cadre pour la Recherche et le Développement technologique). Le développement de l'outil a duré plus de 10 ans.

tant l'utilisateur au cœur de la solution. Nous avons choisi une méthode qui le permet. L'ordinateur et les algorithmes développés ne sont là que pour aider l'utilisateur à prendre les décisions. C'est l'utilisateur qui valide toutes les étapes de la restructuration et qui effectue les corrections d'erreurs au fil de l'eau.

1.4 Plan et contributions

Dans la section 1, nous introduisons les notions de documents, de documents structurés, de documents non structurés, de documents semi-structurés et de documents destructurés, ainsi que les notions de « *big data* » et « *small data* » puis nous répondons aux trois questions : pourquoi ? quels documents ? et comment restructurer les documents déstructurés ?

La section 2, est consacrée à l'état de l'art lié aux travaux de la thèse. Dans cet état de l'art, nous commençons par expliquer pourquoi nous avons écarté les réseaux de neurones comme solution, puis nous regardons ce qui s'est fait dans le domaine de la structuration des documents déstructurés de type PDF en privilégiant cinq domaines : les formules mathématiques, les tableaux et les formulaires, les articles et revues de presse, les articles scientifiques, et, pour terminer les dessins, des schémas, des plans. Nous poursuivons par un état de l'art sur la méthode KDD qui est à la base de la méthode que nous avons développée pour solutionner le problème de la restructuration. Nous faisons un état de l'art sur la création des tableaux de suffixes qui est au coeur de la fouille des données de la méthode KDD. Nous concluons cette section par les outils du marché en montrant leurs limites.

Dans la section 3, nous expliquons les apports de la méthode KDD dans la restructuration des données qui sont matérialisées par les quatre phases de la méthode (A)KDD : 1) la sélection et le nettoyage, 2) la codification, 3) la fouille de données 4) L'interprétation-évaluation des résultats. Puis nous détaillons pourquoi et comment nous avons développé la méthode (A)KDD qui est dérivée de la méthode KDD en ajoutant l'aspect incrémental (indispensable pour la reconstruction de l'information), et l'aspect centré sur l'utilisateur (qui explique le A de Antropocentré de notre méthode (A)KDD. Nous développons la sélection des données d'entrée (phase 1) et comment les nettoyer. Nous expliquons le principe de la codification (phase 2) et nous justifions la nécessité de développer une nouvelle méthode pour trouver les motifs (phase 3, la fouille) qui se répètent à l'intérieur d'une chaîne. Cette méthode repose sur la création du tableau des suffixes, du tableau des LCP (*longest Common Prefix array*) et de la détection des motifs en un seul algorithme que nous avons nommé « SAP ». Nous terminons par l'interprétation-évaluation des résultats (phase4) et la présentation des résultats à l'aide d'une ontologie du domaine.

Dans la section 4, nous détaillons chaque étape incrémentale. Nous décrivons de façon très précise la codification , la fouille et l'interprétation-évaluation pour chacune des étapes incrémentales.

Dans la section 5, nous présentons les résultats obtenus par la méthode

« SAP» pour trouver des motifs et nous la comparons aux derniers résultats récents obtenus par Puglissi sur des chaînes de caractères. Puis nous détaillons les résultats obtenus pour restructurer les documents dans le domaine de la schématique d'automatisme sans pouvoir les comparer à d'autres solutions puisque nous n'en avons pas trouvé.

Dans la section 6, nous concluons cette thèse en montrant les avancées et les limites de la solution de la méthode (A)KDD et nous précisons les perspectives qu'offre la méthode.

Les contributions des travaux développés dans cette thèse sont les suivantes.

C'est la première fois que l'hypothèse de l'ordre chronologique est prise comme principe de base pour restructurer des documents déstructurés. Dans le cas des fichiers PDF créés par des logiciels de DAO/CAO/PAO, nous affirmons que la création des objets graphiques suit toujours la même logique et la même chronologie. Par exemple, quand le logiciel doit inscrire, dans le fichier PDF, le texte « KM12 », il enregistre les lettres dans cet ordre et les lettres sont rangées les unes après les autres. Ce principe est le même si le logiciel enregistre un symbole dans le fichier PDF, les éléments graphiques de ce symbole sont, chaque fois que le symbole apparaît, exactement dans le même ordre. Cette hypothèse est vérifiée dans tous les exemples que nous avons pris (voir page 55). Cette hypothèse n'a pu être prise que parce que l'on a développé les outils qui permettent d'accéder directement au contenu des fichiers PDF (voir paragraphe page 51) sans passer par des outils développés par des tiers qui peuvent modifier l'ordre des objets enregistrés.

La transformation d'un espace 2D, qui est habituellement l'espace dans lequel sont décrits les schémas, les plans ou les dessins, en un espace 1D qui est matérialisé par une chaîne de codes. Cette transformation 2D en 1D n'est possible et n'a un intérêt que grâce à l'hypothèse décrite ci-dessus. La transformation 2D en 1D tient compte de l'élément graphique précédent, d'où l'importance d'avoir un ordre chronologique d'enregistrement qui soit respecté lors de l'apparition du même élément (voir paragraphes pages 67, 74, 76, 86, 88, 92, 95) .

L'adaptation de la méthode KDD à la fouille de données graphiques alors que cette méthode était initialement orientée base de données afin de trouver les règles d'apparition de répétitions des données dans ces bases (section 3.1). L'ajout de l'aspect incrémental (section 3.2) à la méthode KDD et l'ajout de l'aspect antropocentré sur l'utilisateur (section 3.3), nous ont permis de transformer la méthode KDD en méthode (A)KDD, et de l'utiliser pour la fouille des données graphiques. L'aspect incrémental diminue le volume des données à traiter (c'est une préoccupation permanente de ces travaux) en remplaçant x données graphiques par un élément structurant de plus haut niveau. L'aspect centré utilisateur nous permet d'avoir un contrôle permanent de l'utilisateur et être sûr, à 99,99%, que la restructuration est conforme à ce que veut l'utilisateur.

Une méthode de nettoyage et de préparation des données d'entrée complète et efficace qui est une condition nécessaire pour la bonne analyse des données par la suite.

Le développement d'une nouvelle méthode de détection des motifs à l'intérieur d'une chaîne (section 3.6). Méthode qui fusionne en un seul algorithme plusieurs opérations qui, dans de nombreux travaux de recherches, se font séquentiellement. Cette fusion diminue les temps de recherche de façon significative grâce à deux paramètres qui fixent le nombre de répétitions recherchées et la longueur des répétitions.

La définition des méthodes à mettre en œuvre à chaque incrémentation pour coder l'information, fouiller l'information, la classer et la présenter à l'utilisateur ceci grâce à une ontologie adaptée au domaine traité et que l'utilisateur peut adapter à ses besoins.

Section 2

État de l'art

L'état de l'art de la restructuration des documents déstructurés vectoriels de type graphique est quasiment inexistant. Depuis les années 2000, beaucoup d'équipes se sont penchées sur la restructuration des documents déstructurés, comme les fichiers PDF, mais sans traiter la partie graphique.

La question suivante nous a souvent été posée : pourquoi n'utilisez-vous pas les réseaux de neurones à convolution pour restructurer les documents déstructurés ?

Nous commençons donc notre état de l'art sur les réseaux de neurones à convolution.

2.1 Les réseaux de neurones à convolution

Notre réponse est que les réseaux de neurones ne peuvent pas répondre à notre problème pour 4 raisons : la difficulté et le coût de créer les jeux de données d'apprentissage et de tests, le choix et la taille du réseau de neurones, le temps et le coût des calculs, la taille des images en entrée.

2.1.1 La difficulté de créer les jeux de données

Prenons les travaux de Yann LeCun en 1998 [LeC+98]. Ces travaux portent sur l'identification de chiffres ou de nombres manuscrits comme les codes postaux ou les montants des chèques. En entrée, il a l'image du chiffre ou du nombre manuscrit, et en sortie il produit ce chiffre ou ce nombre sous forme numérique. Il solutionne la reconnaissance des 10 chiffres avec un réseau de neurones à convolution de nom « leNet-5 ». Ce réseau à convolution est composé de 7 couches (non compris l'entrée). Il contient 340 908 connections entre les neurones et seulement 60 000 paramètres car certains poids sont partagés par plusieurs connections. Les jeux de données pour entraîner le réseau sont composés de 60 000 chiffres issus de la base du MIT qui a été écrite par 500 personnes, et complétés de 540 000 de ces mêmes images avec de légères distorsions, calculées informatiquement.

Ils ont disposé d'environ 600 000 données de tests pour déterminer les 60 000 paramètres qui permettront de reconnaître les 10 chiffres. Ce que l'on peut énoncer différemment, pour la reconnaissance d'un chiffre : il faut 6 000 représentations de base différentes et 60 000 représentations déduites de ces 6000 représentations de base.

Dans le domaine de la schématique électrique, il existe environ 400 symboles principaux. Ce qui veut dire, que si l'on respecte les mêmes ratios que Yann LeCun, il faudrait créer une base de tests composée de 2 400 000 représentations de base, provenant de plusieurs logiciels différents, et de 24 000 000 déduites de ces données de base. Ceci est impossible à réaliser, à notre avis, pour une PME et même un grand groupe. Impossible pour deux raisons suivantes. La première, c'est qu'il est impossible d'obtenir les 2 400 000 représentations de base provenant de plusieurs logiciels différents. La deuxième, c'est que même si l'on obtient ces 2 400 000 représentations et que l'on évalue à 30 secondes le temps nécessaire pour extraire un symbole, l'identifier et lui donner son étiquette (ce qui est très peu), cela prendrait à une personne qui travaille 24 heures sur 24, et 7 jours sur 7, environ 23,5 années. Ceci uniquement pour la schématique électrique, mais les chiffres sont du même ordre pour la schématique hydraulique, pneumatique, climatique, électronique, du bâtiment, etc. Seules les très grandes entreprises comme les GAFA, peuvent disposer de ce volume de données et peuvent créer ces jeux de données.

2.1.2 Le choix et la taille du réseau de neurones

Le réseau de neurones de Yann LeCun serait sûrement insuffisant pour classifier les 400 symboles. Il faudrait prendre les technologies plus récentes développées pour :

— AlexNet en 2012 [KSH12]
— VGG16 en 2014 [SZ14]
— GoogleLeNet (Inception V1) en 2014 [Sze+14]
— ResNet en 2015 [He+16]
— Inception-ResNet (Inception V4) en 2017 [Sze+17],
— SE-RESNET en 2018 [HSS18]
— NASNet en 2019 [Zop+18].

Pour ces réseaux de neurones, le nombre de paramètres varie entre 134 000 000 pour « VCG » (soit 2 000 fois plus que le nombre de paramètres de « LeNet-5 » et quelques millions pour« NasNet » (entre 3 100 000 et 37 000 000 suivant la version de « NasNet »). Ce qui veut dire un jeu d'apprentissage encore plus grand que celui de « LeNet-5 » (pour rappel Yann LeCun utilisait 1 représentation de base et 10 déduites pour 1 paramètre).

Aujourd'hui, la base « ImageNet » qui sert d'apprentissage pour ces réseaux de neurones contient près de 15 millions d'images annotées à la main. Ces images sont réparties dans plus de 21 000 groupes ou classes (en anglais : *synsets*). Plus d'un million contiennent la boîte englobante autour de l'objet. Ce n'est plus 60 000 chiffres comme pour « leNet-5 » mais des millions d'images qui sont nécessaires pour entraîner ces réseaux. Là encore, ce n'est pas accessible à une

entreprise de type PME ou ETI et probablement pas par un grand groupe hors GAFA.

2.1.3 Le temps et le coût de calcul

À titre indicatif, pour « Inception V4 » [Sze+17] le temps d'apprentissage avec 800 GPU (*Graphics Processing Unit*) est de 28 jours, soit 22 400 jours GPU, ou encore, 537 600 heures GPU. Si on valorise le coût d'apprentissage avec un prix d'environ 2 € de l'heure, cela représente plus de 1 000 000 €. Un tel coût est prohibitif pour de nombreux laboratoires et, à plus forte raison, pour des PME. D'autant plus que ce temps correspond uniquement à un, et un seul, apprentissage. En général, il faut plusieurs dizaines, voire centaines d'apprentissages différents pour structurer et régler correctement le réseau de neurones (nombre de couches, enchaînement des couches, etc).

2.1.4 La taille des images

Dans « leNet-5 » la taille des images qui matérialisent les chiffres est de 32*32 pixels, ce qui est très bien pour matérialiser tous les chiffres ; de plus tous les chiffres ont la même taille. Pour les symboles électriques c'est différent. Tous les symboles ont des tailles différentes. Le plus petit a une taille 20*20 pixels (taille proche de 2 mm*2 mm en résolution 300 DPI), 75% des symboles ont des tailles comprises entre 200*200 pixels et 500*500 pixels (ce sont tous les symboles électriques normalisés), ensuite environ 20% ont des tailles proche de 1000*2000 pixels (ce sont les cartes d'automates, les variateurs et tous les types d'équipements propres à chaque constructeur), et les 5% restants sont les cartouches qui ont une taille de 3000*4200 pixels.

Les derniers réseaux de neurones (comme « NasNet ») prennent en entrée des Images de 224*224 pixels, ce qui multiplie par presque 49 la surface des images de « LeNet-5 », mais qui reste encore insuffisant par rapport à la taille des cartouches qui est 250 fois plus grande que la surface des images de « NasNet ». Ce sont encore des points très contraignants pour utiliser les réseaux de neurones à convolution.

2.1.5 Des solutions alternatives

Des solutions alternatives commencent à se développer pour essayer de contrer cette problématique du nombre de paramètres du réseau de neurones et de la taille du jeu de donnés d'apprentissage. Par exemple, en 2019, dans le domaine de l'analyse et de la structuration des documents historiques, dans « *A Comprehensive Study of ImageNet Pre-Training for Historical Document Image Analysis* » Studer et al. [Stu+19], les auteurs (Linda Studer, Michele Alberti, Vinaychandran Pondenkandath, Pinar Goktepe, Thomas Kolonko, Andreas Fischeryz, Marcus Liwickiyx, Rolf Ingold) considèrent qu'« un moyen prometteur de faire face au manque de données d'apprentissage est d'en-

traîner les réseaux de neurones sur des images d'un domaine différent ». Ils constatent que « l'utilisation d'ImageNet a un effet positif sur la classification, mitigé sur l'analyse sémantique, et même parfois nuisible pour certains réseaux de neurones ». Ils concluent l'article sur le fait que « dans l'analyse des images de documents historiques, le manque de données d'entraînement annotées est souvent l'un des facteurs les plus limitatifs pour l'apprentissage automatique ».

Dans cette conclusion, nous retrouvons une des raisons pour laquelle nous n'avons pas choisi les réseaux de neurones à convolution pour restructurer les documents déstructurés de type graphique.

2.2 La restructuration des fichiers PDF

La restructuration des fichiers PDF, et des informations contenues dans ces fichiers, ont entraîné de nombreux travaux qui ont souvent trouvé leur origine dans la restructuration des informations qui étaient présentes sur les documents papier, et l'on retrouve cette influence dans certains travaux où l'image TIFF (*Tagged Image File Format*) des pages est utilisée.

Les premiers travaux que nous allons citer sont relatifs à la restructuration des formules mathématiques car ces formules contiennent des éléments graphiques et des textes. Puis, nous aborderons les travaux sur les tableaux (de type Excel) contenus dans les fichiers PDF, car ils comportent également des éléments graphiques : en général, des segments de droite, des rectangles et, bien sûr, du texte. Ensuite, nous regarderons ce qui s'est fait dans le domaine de la restructuration des articles de presse et, de façon plus générale, sur la restructuration des documents contenant du texte et des images, et, plus rarement, des éléments graphiques. Nous compléterons par les travaux assez récents qui sont en cours sur les articles scientifiques stockés au format PDF, car il devient presque impossible de les traiter manuellement, vu leur nombre. Et nous terminerons par le seul travail qui correspond à notre domaine : la restructuration des documents graphiques de type schéma électrique.

2.2.1 Restructuration des formules mathématiques

En 2012, dans « *MaxTract : Converting PDF to LATEX, MathML and Text* » ([Bak12]) Josef B. Baker, Alan P. Sexton, et Volker Sorge analysent l'image au format TIFF du fichier PDF d'entrée. De cette façon, ils localisent les boîtes englobantes des formules mathématiques contenant des symboles mathématiques (glyphes) et des composants liés (lettres, chiffres, caractères spéciaux). À leur avis, cela est nécessaire car le format PDF ne code pas les informations de la boîte englobante, ce qui est exact, et qui pour eux est indispensable pour l'analyse et la reconnaissance de formules mathématiques. Par contre, ils auraient pu calculer cette boîte englobante directement avec les objets du fichier PDF qui contiennent les coordonnées X et Y qui permettent de calculer la boîte englobante de l'objet graphique. Mais, comme nous l'avons dit, en introduction, certains travaux réutilisent les connaissances acquises lors de la restructuration

des documents papier. Après cette localisation, les boîtes englobantes sont mappées avec les informations de caractères et de polices extraites via l'analyse PDF pour produire une liste d'éléments avec leur emplacement détaillé et leur police. La coupe du profil de projection de ces éléments est ensuite utilisée pour identifier les lignes d'éléments qui sont passées à une grammaire linéaire pour analyse.

En 2012, dans « *Recognition and retrieval of mathematical expressions* » ([ZB12]) puis en 2014, dans « *Processing Mathematical Notation* » ([BZ14]) Richard Zanibbi et Dorothea Blostein constatent que la reconnaissance et la récupération de graphiques tels que des images, des figures, des tableaux, des diagrammes et des expressions mathématiques, sont à des stades relativement peu avancés. Ils présentent quatre problèmes clés de reconnaissance des formules mathématiques (la détection des expressions, la classification des symboles, l'analyse de la disposition des symboles et la construction d'une représentation de ces expressions). L'apprentissage automatique qu'ils proposent, consiste à optimiser les algorithmes du système de reconnaissance des formules mathématiques et à développer des algorithmes efficaces d'indexation avec l'aide de l'utilisateur. Un de leurs problèmes concerne le développement d'interfaces utilisateur intégrant de manière transparente la reconnaissance et la récupération des formules. Dans leur conclusion, ils abordent deux points. Le premier concerne l'augmentation de l'activité dans ce domaine de recherche car les problèmes de reconnaissance des formules mathématiques sont communs à de nombreuses applications de récupération de documents. Le deuxième est relatif aux futurs développements qui devront mettre à disposition des utilisateurs des interfaces simples en relation avec les algorithmes développés.

En 2017, dans sa thèse « *Augmenting Mathematical Formulae for More Effective Querying & Efficient Presentation* » ([Sch17]) Moritz Schubotz étudie les possibilités et les limites du produit « *MaxTract* ». Pour Schubotz le fait que le produit ne traite que les fichiers PDF contenant des fontes de type 1 est très limitatif. La représentation des fontes de type 1, n'est qu'une des 4 possibilités d'utiliser les fontes dans un fichier PDF. C'est un point important qui sera détaillé quand nous traiterons les textes (caractères, mots, phrases, paragraphes).

Dans notre cas, pour l'analyse des schémas électriques, les formules sont peu nombreuses et même quasi inexistantes. Le seul cas que nous avons à traiter est, par exemple, « m^3/h ». Mais ceci est plus un problème d'analyse de texte que de formule mathématique. Si une formule mathématique est présente dans le document, elle sera considérée comme un ensemble de symboles et de lettres et son encombrement sera directement calculé à partir des éléments graphiques contenus dans le fichier PDF, contrairement à la méthode développée pour « *MaxTract* » ([Bak12]). L'utilisateur aura la possibilité de grouper l'ensemble des symboles et des composantes textuelles (lettres et chiffres), de cette façon ils formeront un tout, un peu comme une image. La conclusion qu'ils font, et que certains chercheurs dans le domaine de la reconnaissance des formules mathématiques [YF04] font également, porte sur le fait qu'il est nécessaire de développer des interfaces simples pour permettre à l'utilisateur d'interagir avec le système, et, que cette nécessité a des répercussions sur les algorithmes déve-

loppés. C'est une conclusion à laquelle nous adhérons et qui fait l'objet de cette thèse. Il faut que les algorithmes développés soit compatibles avec des temps d'attente acceptables par l'utilisateur.

2.2.2 Restructuration des tableaux

Parmi les premiers travaux sur ce sujet, on peut citer, en 2003, l'article « *Detection, extraction and representation of table* » ([Ram+03]) de Jean-Yves Ramel et de Nicole Vincent. Ils présentent une double méthode d'extraction des tables contenues dans les fichiers PDF. La première méthode est relative aux éléments graphiques, segments de droite, rectangles, polylignes, qu'ils décomposent en segments de droite, puis ils déterminent la boîte englobante de tous les segments de droite ayant un point commun.

Mais ils indiquent que la structure finale du tableau ne peut être extraite de cette partie graphique : il faut analyser les textes plus en détail, ce qui est l'objet de leur deuxième méthode. Cette deuxième méthode consiste à recréer des blocs de textes à partir des éléments textuels présents dans les fichiers PDF. Puis, à l'aide d'une analyse descendante, ils commencent par traiter la configuration globale de ces blocs de textes. Ensuite, ils prennent en compte les caractéristiques locales de chaque bloc pour former des colonnes, et, ils terminent par l'étude des lignes des tableaux. Ils enregistrent l'ensemble de ces blocs dans un graphe orienté (les nœuds sont les blocs, les arcs les relations) et déterminent ainsi les cellules du tableau. Ils concluent par le fait que ces méthodes peuvent être approfondies et raffinées afin de reconstruire la structure physique et logique des tables complexes, et que l'identification des éléments graphiques doit être combinée à l'information textuelle, ce qu'ils n'ont pas fait. Cette combinaison textes et graphiques est fondamentale pour nous. Beaucoup de travaux ne traitent que le texte, mais, pour la restructuration des schémas électriques, nous devons traiter simultanément le texte et le graphique.

En 2007, dans « *Table Recognition and Understanding from PDF Files* » ([HB07]), Tamir Hassan et Robert Baumgartner compléteront les travaux de Ramel et Vincent. Ils commencent par décrire les étapes pour transformer les instructions PDF de bas niveau en blocs de texte et lignes, pour cela, ils utilisent l'outil « *PDFBox Java library2* ». Puis, ils proposent trois différentes classifications pour les tables. Ils développent des méthodes pour détecter ces tables et identifier correctement leurs lignes et colonnes. Ils utilisent le « Principe de confinement rectangulaire » qui repose sur la position des textes. Ils n'utilisent pas les cadres qui pourraient exister dans les tableaux car leur solution doit fonctionner pour des tableaux avec ou sans cadres et la majorité de leurs exemples de tableaux à restructurer sont des tableaux sans cadres.

En 2009, dans « *User-Guided Wrapping of PDF Documents Using Graph Matching Techniques* » ([Has09], Tamir Hassam met l'utilisateur au cœur du problème de la reconstruction des tableaux contenus dans les fichiers PDF. Il développe une interface graphique qui permet de faire correspondre l'arbre obtenu par la restructuration des tableaux avec la réalité physique du tableau sur la page. Cette correspondance se fait sous le contrôle de l'utilisateur.

En 2012, vu le nombre de plus en plus important de travaux, M Göbel, T Hassan, E Oro et G Orsi, proposent une méthodologie pour évaluer les outils issus des travaux de recherche dans « *A methodology for evaluating algorithms for table understanding in PDF documents* » ([Gob+13]). Une méthodologie équivalente devra être développée dans le domaine de la restructuration des documents graphiques.

En 2016, dans « *Configurable table structure recognition in untagged PDF documents* » ([SMA16]), A. Shigarov, A.Mikhailov et A. Altaev proposent une méthode qui repose sur 9 heuristiques, et, qui, à l'aide de 4 paramètres, permet de structurer les tableaux contenus dans les fichiers PDF. Ils concluent l'article par la nécessité d'augmenter le nombre d'heuristiques, par la nécessité d'améliorer les prétraitements et le nettoyage des données en entrée, et, par la nécessité de détecter automatiquement la position des tableaux dans les fichiers. En 2017 dans « *Rule-based spreadsheet data transformation from arbitrary to relational tables* », Shigarov ([SM17] aborde la transformation de données issues des tableurs sous une forme canonique (relationnelle) et il veut l'appliquer à la restructuration des tables issues des fichiers PDF.

En 2019, dans « *Tablepedia : Automating PDF Table Reading in an Experimental Evidence Exploration and Analytic System* » ([Yu+19]), Yu, Li, Zeng et Jiang proposent de lier les tableaux contenus dans les PDF au texte lui-même contenu dans les fichiers PDF. L'idée est très originale, surtout que leur objectif est de permettre une lecture plus facile des PDF contenant des tableaux. Ils développent une interface utilisateurs qui met en évidence les cellules des tableaux avec le texte présent sur le document. Dans les schémas électriques, les tableaux sont peu nombreux mais ils sont présents. Par exemple, dans un dossier, il peut exister des folios qui contiennent la liste des liaisons (équipotentielles). On peut également avoir des folios qui contiennent la liste du matériel utilisé dans le dossier. Ou encore avoir la liste des folios avec leur sommaire et sous sommaire. Mais dans le cas des schémas électriques, ces tableaux font l'objet de cartouches spécifiques (et donc de symboles prédéterminés) qui sont en général partiellement remplis. Pour notre cas d'usage, l'identification des tableaux dans un dossier électrique revient à identifier les symboles qui matérialisent ces tableaux. Les textes contenus dans les cases des tableaux peuvent être laissés libres ou regroupés en un symbole. Le point important de ces derniers travaux sur les tableaux montre aussi la nécessité de mettre l'utilisateur au cœur du système. Dans les plus anciens travaux, les solutions sont conçues pour fonctionner automatiquement sans l'intervention de l'utilisateur, si ce n'est par le réglage d'un ou deux paramètres. Mais, vu la diversité des tableaux, les dernières recherches ([Yu+19]) impliquent l'utilisateur dans la restructuration des tableaux.

2.2.3 Restructuration des articles de presse ou de revues

En 1995, dans l'article « *Document analysis of PDF files methods, result and implication* » ([LB95]), William S. Lovegrove et David F. Brailsford décrivent une stratégie d'analyse de documents de type PDF contenant des articles de presse. Cette stratégie examine l'apparence et la position géométrique

des blocs de texte et des images présents sur un document. Un système de tableau noir est utilisé pour marquer les blocs et déduire les relations fondamentales qui existent entre eux. L'analyse du PDF donne une décomposition de la page avec précision. Les auteurs utilisent les outils d'Adobe™ pour extraire les textes et ils constatent un nombre important d'erreurs de ces outils quand les documents sont sur plusieurs colonnes et que l'espace entre les colonnes est faible. Ils constatent également de nombreuses anomalies quand le document a été tourné à 90°.

Durant la même année, Philip N. Smith et David Brailsford travaillent sur les fichiers PDF en utilisant, comme base de travail, la gamme de produits Acrobat qu'ils présentent en détail dans « *Towards structured, block-based PDF* » ([SB95]). Un accent particulier est mis sur la structure orientée objet, qui, à leur avis, n'a pas encore été pleinement exploitée. Ils constatent que les outils, proposés dans Acrobat, permettent seulement de représenter une série de pages et des ressources associées et qu'aucune structure logique n'est possible. Pour combler ce vide, ils proposent une définition d'un PDF encapsulé (EPDF), dans laquelle les blocs comportent avec eux leurs propres besoins en ressources, ainsi que des informations géométriques et logiques. Ils décrivent un formateur de blocs qu'ils appellent « *Juggler* ». Ce formateur crée les blocs et associe les ressources des pages. Les auteurs concluent en disant que le mécanisme EPDF – Juggler fonctionne bien, mais qu'il existe certains problèmes de mise en œuvre (des problèmes de nommage des blocs, de déclaration des ressources au niveau de la page...), et des obstacles plus graves pour poursuivre ces développements comme l'impossibilité de définir la longueur d'une ligne ou d'une section (« *impossible to reformat for a different line length (measure) or to adjust section* »).Dans notre cas, après reconstruction des mots ou des phrases, les logiciels de CAO/DAO/PAO permettent tous de justifier correctement tous les textes.

En 2003, dans « *Creating reusable well-structured PDF as a sequence of component object graphic (COG) elements* » [BBH03], Steven R. Bagley remarquent que, bien que chacune des pages dans un document PDF soit un objet graphique indépendant, cette propriété ne s'étend pas nécessairement aux composants (en-têtes, diagrammes, paragraphes, etc.) dans une page. Ceci rend très difficile et incertaine la manipulation et l'extraction d'objets graphiques sur une page PDF. Ils étudient les avantages d'un modèle dans lequel les pages PDF sont créées à partir d'assemblages de COG (*Component Object Graphics*) chacun avec un graphisme clairement défini. Le positionnement relatif des COG sur une page PDF est déterminé par des objets qu'ils appellent «spacer» et, en parcourant l'arbre des COG, ils obtiennent le rendu voulu. En conclusion, ils espèrent, qu'après avoir acquis un peu plus d'expérience dans la génération et la manipulation des fichiers COG-PDF, ils s'attaqueront à la tâche beaucoup plus difficile d'essayer de réécrire des pages PDF en séquences de COG. Ce qui, à notre connaissance, n'a pas été fait.

Bien qu'anciens, ces 3 travaux sont très intéressants car ils n'utilisent que les informations contenues dans les fichiers PDF, mais leurs objectifs sont différents des nôtres. Ils veulent compléter le contenu des fichiers PDF, et ils se sont heurtés aux possibilités limitées de la syntaxe du PDF, ce qui ne sera pas

notre cas, car les logiciels vers lesquels nous exportons nos données offrent plus de possibilités graphiques que le PDF. Ils constatent que, en utilisant les outils d'Abobe™, ils ont des problèmes de mauvaise association des lettres et des mots. C'est, en général, le même constat que font tous les travaux de recherche qui utilisent des outils externes. Ces outils externes font des prétraitements de l'information qui ne sont pas toujours corrects. C'est pourquoi, nous avons développé nos propres outils de lecture des fichiers PDF. Ces outils ont été développés pendant les 2 ans de maturation du projet de réalisation de la thèse. Sans ces outils, qui sont hors cadre des travaux de la thèse, puisque essentiellement des outils d'ingénierie, nous n'aurions pas pu développer la méthodologie contenue dans cette thèse.

En 2006, dans « *A System for Converting PDF Documents into Structured XML format* » [DM06]), Dejean et Meunier ont deux objectifs : a) reconstituer les mots en associant les lettres, puis les lignes, les paragraphes et les articles, b) séparer les textes des images ou zones vectorielles.

Les premières étapes consistent à extraire du fichier PDF les informations de base : textes, vecteurs, polylignes et objets externes. Puis ils calculent l'organisation textuelle de chaque page et son ordre de lecture à l'aide d'une découpe descendante (*Top Down*) qui découpe l'espace XY. De cette façon, ils extraient la structure logique du document. Ils détectent, également, la table des matières du document et ils la structurent en tenant compte des niveaux hiérarchiques. Ils présentent également une interface graphique qui permet à l'utilisateur de corriger chaque étape de la conversion.

En 2009, dans « Restructuration physique et logique de documents électroniques » [BI], Jean-Luc Blochel et Rolf Ingold proposent une approche différente et efficace pour reconstruire la structure physique et logique des fichiers. Le point très intéressant de leur démarche est d'élaborer un système d'analyse de documents électroniques textuels qui fonctionne aussi bien pour un document textuel, quelle que soit son origine (Word™, Excel™, PowerPoint™, etc ...), sur support papier, que pour un document au format PDF. Pour arriver à ce résultat, ils utilisent tout le savoir-faire qu'ils ont pu acquérir dans le cadre du projet CIDRE pour Coopérative & Interactive Document Reverse Engineering, voir la thèse de Lyse Robardey à ce sujet ([Rob01]). Ils transforment les pages PDF du document en image TIFF. De cette façon, ils peuvent appliquer toutes les solutions qui ont été développées dans le cadre du traitement de l'image. Ils sont convaincus que l'extraction des primitives d'un fichier PDF conjointement à l'application d'algorithmes classiques d'analyse de l'image d'un document, permettront, non seulement, de reconstituer la structure physique du document, mais aussi sa structure logique. Cette solution a un gros avantage car elle permet de traiter les documents papier et PDF de la même façon. Dans le cas de la présence d'un fichier PDF, ils mettent en concordance la structure des objets graphiques qu'ils ont identifiés dans le fichier PDF avec les informations de structure logiques qu'ils ont obtenues à partir de l'analyse de l'image. La raison qu'ils donnent, pour procéder de cette façon, c'est que la séparation en zones de texte et en zones graphiques, tâche relativement aisée en appliquant des méthodes d'analyse d'image, peut s'avérer très compliquée en utilisant uniquement l'information contenue dans le PDF, surtout lorsqu'une

zone graphique est constituée d'éléments textuels. Les expériences qu'ils ont pu faire, sur la structure des fichiers PDF, leur ont montré que cela génère de grandes difficultés parce que les informations structurelles ne sont pas toujours fiables. Ils disent que, dans des documents multi-colonnes, l'ordre d'apparition des blocs de texte ne reflète en général pas l'ordre de lecture. Pire, il arrive que des portions de phrase ou des mots isolés n'apparaissent pas dans leur contexte mais de manière isolée à la fin d'un fichier. Ils concluent que ce type d'imperfection dépend de l'historique du document et des logiciels qui ont servi à le produire.

Leur conclusion est très juste, cela dépend de l'historique du document et des logiciels qui l'ont produit. Dans leurs travaux, ils utilisent les unes des journaux comme support de leur recherche. Bien que produites par des outils de PAO, les rédacteurs ont la possibilité de « forcer » certaines parties de la mise en page pour gagner du temps. Dans notre contexte, c'est beaucoup moins le cas car les informations dans un logiciel de CAO doivent respecter une structure bien précise, c'est de cette structure précise que sont déduites des informations. Ce n'est pas forcément le cas pour les logiciels de DAO qui peuvent offrir une certaine souplesse.

À notre avis, il est dommage de ne pas utiliser le fait que les informations créées dans un fichier PDF le sont en respectant un ordre très logique. Dans notre cas, c'est vrai dans presque cent pour cent des cas. Ce n'est pas parce qu'il y a quelques cas particuliers qu'il ne faut pas utiliser cette caractéristique vraie dans plus de 99% des objets contenus dans les fichiers PDF que nous aurons à restructurer. Il faudra traiter les quelques cas particuliers sans tenir compte du fait que les informations sont créées dans un ordre chronologique.

Les mêmes auteurs ont ajouté, en 2006, dans « XCDF : Un format canonique pour la représentation de documents» [Rig+04], un outil interactif d'apprentissage, DOLORES (*DOcument LOgical REStructuring*), pour la restructuration de la logique du document. DOLORES met l'utilisateur au cœur du système. C'est lui qui crée un modèle par interaction, apprentissage et correction.

2.2.4 Restructuration des articles scientifiques

Concernant la restructuration des articles scientifiques, nous constatons, depuis quelques années, une très forte activité dans ce domaine. De nombreuses équipes ont essayé de restructurer et d'extraire un maximum d'informations de ces articles voir [RI] et [SGF15].

En 2017, dans « *Automatically Determining Versions of Scholarly Article* » [RS17] DH Rothchild et SM Shieber décrivent un système qui reçoit un ensemble de documents au format PDF, et génèrent une base de connaissances représentée par un graphe. Le système est composé de trois modules principaux : un analyseur, un extracteur d'entité-relation, et un constructeur de taxonomie graphique. En 2018, dans « *Extracting semantic relations for scholarly knowledge base construction* » RA Al-Zaidy et CL Giles analysent les fichiers PDF contenant des articles scientifiques pour établir le lien qui existe

entre eux. Leur méthode extrait la sémantique des articles en tant que concepts et qu'instances. Les instances contiennent le texte intégral du document. Ils extraient deux types de relations entre les concepts, en utilisant un algorithme d'apprentissage itératif.

2.2.5 Restructuration des dessins, des schémas, des plans

Déjà, en 2003, l'équipe du LORIA de Nancy, en particulier, Karl Tombre et Bart Lamiroy, faisait le constat suivant [TL03] : « *The grassroot user, as well as large companies, have a huge amount of information at their disposal, but this information is available in very "poor" formats : paper documents, or low-level, poorly structured digital formats such as Postscript, PDF or DXF. The challenge is therefore to convert this poorly structured information into enriched information which can be used within an information system* ». Ce constat sur « les formats numériques mal structurés » et sur « le defi qui consiste donc à convertir ces informations mal structurées en informations enrichies » a été fait pendant la conférence ICDAR de 2003, mais les chercheurs qui travaillaient sur les documents papier ne se sont pas intéressés aux informations contenues dans les fichiers vectoriels que Karl Tombre et Bart Lamiroy décrivent : PostScript, PDF ou DXF.

Á notre connaissance, il existe une seule équipe de recherche qui se soit intéressée à l'extraction des connaissances dans les documents très graphiques comme les schémas électriques. C'est l'équipe Mnémosyne de l'INRIA de Bordeaux. Elle aborde ce problème dans l'article d'Ikram Chraibi Kaadoud en 2016 « *Implicit knowledge extraction and structuration from electrical diagrams* » [KRA17]. Ikram Chraibi Kaadoud extrait tous les textes des fichiers PDF. Puis elle utilise des méthodes inspirées de l'ECD (Extractions des Connaissances des Données) ou KDD en anglais (*Knowledge Discovery in Database*), pour obtenir les informations nécessaires à la structuration des connaissances. Connaissances qu'elle modélise avec des dendrogrammes.

Ces méthodes et ces solutions sont développées dans sa thèse de 2018 [Kaa18] ayant pour titre « Apprentissage de séquences et extraction de règles de réseaux récurrents : application au traçage de schémas techniques ». Dans son travail, elle n'utilise que les informations textuelles qu'elle extrait avec des outils en python. Elle laisse de côté toute la partie graphique du dossier. Son objectif est de retrouver la structure d'un dossier. Elle le fait en utilisant uniquement les textes et les notions de renvoi de folio qu'elle identifie dans les textes. Cette solution permet de naviguer de page en page et de trouver ainsi la structure du dossier.

Un point important que nous retiendrons de ces travaux est l'utilisation des méthodes fortement inspirées de l'ECD (ou KDD) que nous présentons ci-dessous.

2.3 La méthode KDD

Dans la méthode proposée par Usama Fayyad en 1996, dans l'article « *The KDD process for extracting useful knowledge from volumes of data* » [FPS96b] mais aussi dans l'article « *From Data Mining toKnowledge discovery in DataBase* » [FPS96a], il décrit les cinq phases fondamentales pour fouiller les données dans une base de données. Ce sont ces cinq mêmes phases : *Selection, Pre-Processing, Tranformation, Data Mining, Interprétation/Évaluation* que nous utilisons (voir figure 2.1) dans notre méthode pour restructurer des éléments de type graphique. Ces phases sont décrites dans le schéma ci-dessous, issu de l'article de Usama Fayyad [FPS96b].

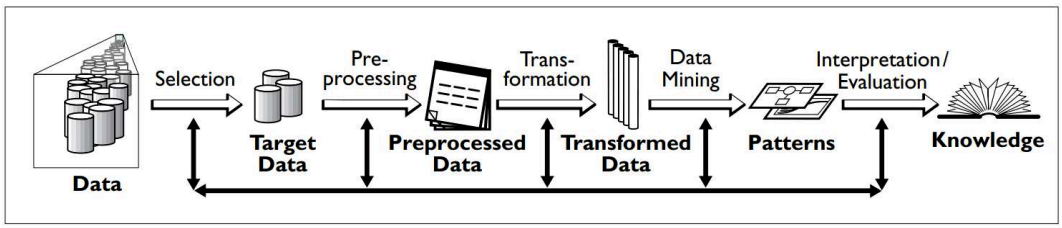


FIGURE 2.1 – Les phases de la méthode KDD de Usama Fayyad en 1996, extrait de « *The KDD Process for Extracting Useful Knowledge from Volumes of Data* » Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth

2.3.1 L'utilisateur absent de la méthode

En 1996, dans les deux articles cités ci-dessus, Usama Fayyad souligne que l'extraction des connaissances doit se faire avec l'aide de l'utilisateur. Lors de la conférence KDD de 2012, dans « *Report on the SIGKDD-2002 panel the perfect data mining tool : interactive or automated ?* » Ankerst reprend les propos de Gregory Piatetsky-Shapiro (co-fondateur de la conférence KDD avec Usama Fayyad) « *The human eye is an excellent tool for spotting natural patterns* » et « *We argue that the goal should be to simplify visualization as much as possible to help humans make more accurate decisions* » [Ank02]. Ce rapport insiste sur le rôle primordial de l'homme dans l'extraction des connaissances des bases de données, mais aussi sur le fait qu'il faut développer des outils qui permettent à l'homme de prendre des décisions simplement, rapidement et avec le plus de précision possible.

Lors de la conférence KDD de 2003, Usama Fayad [FPU03], constate également (et de nouveau) que beaucoup de travaux sur les KDD se font sans l'intervention de l'utilisateur ou, du moins, son intervention ne se fait que dans la phase finale.

C'est le même constat que font Chevrin et les auteurs de « Recherche anthropocentrée de règles d'association pour l'aide à la décision » en 2007 [Che+07]). Ils appuient, encore une fois, sur le rôle de l'utilisateur et c'est l'une des premières fois que l'on voit le terme « anthropocentré » prendre autant d'importance dans l'ECD.

C'est le constat que nous faisons également. En effet, dans la plupart des travaux que nous avons cités pour la restructuration des documents déstructurés, l'utilisateur n'est présent qu'à la fin du processus pour la validation. Ces travaux ne sont pas « anthropocentrés » sur l'utilisateur.

De très nombreux travaux sur l'extraction des connaissances ont pour objectif d'automatiser (cf. rapport de KDD 2002 [Ank02]) au maximum cette extraction. C'est peut-être possible quand les données à traiter sont très nombreuses (*big data*) mais dans notre cas (*small data*) c'est impossible. L'utilisateur qui veut restructurer ces données ne dispose pas de suffisamment de données pour une automatisation complète et fiable du processus de restructuration. Certains travaux que nous avons mentionnés développent de plus en plus d'interfaces avec l'utilisateur, mais ces initiatives sont encore rares.

D'une façon générale, dans le « *Smart Factory*[1]» c'est l'utilisateur qui doit piloter la restructuration avec l'aide de l'ordinateur, ceci pour des raisons de coûts, d'efficacité et de non rejet du système.

2.3.2 L'aspect incrémental inexistant

Avant de parler de l'aspect incrémental, nous allons définir le mot incrémental dans l'extraction des données. Nous utilisons l'adjectif incrémental pour décrire un ajout par palier, petit à petit, afin d'être certain que chaque restructuration ajoutée apporte une amélioration sans créer de dysfonctionnement.

En 2004, dans « *IncSpan incremental mining of sequential patterns in large database* », Chang développe un algorithme efficace, « *IncSpan* », pour la gestion incrémentale d'extraction de motifs [CYH04], car il s'est aperçu qu'un processus incrémental de recherche des motifs (*pattern*) séquentiels était plus efficace pour les extraire des très grandes bases de données.

En 2010, dans « Classification incrémentale supervisée : un panel introductif », Salperwyck [SLB10] aborde ce problème de l'aspect incrémental de la fouille. Son article a pour but de présenter les principales approches de classification supervisées incrémentales recensées dans la littérature. On voit bien que la classification nécessite, dans certains cas, de procéder de façon incrémentale pour fouiller les données et c'est exactement le cas de la fouille des éléments graphiques.

Et, enfin, en 2019, lors de la conférence KDD de Londres, en présentant l'article « *Adaptive Deep Models for Incremental Learning : Considering Capacity Scalability and Sustainability* » [Yan+19], Yand et les auteurs de l'article développent l'idée de la nécessité de concevoir des méthodes incrémentales pour rendre les réseaux de neurones encore plus performants, et, ils proposent la méthode IADM (*Incremental Adaptive Deep Model*).

On voit bien que cet aspect incrémental est une nécessité dans la fouille des données. C'est un point important que nous développons dans notre méthode (A)KDD en même temps que l'aspect anthropocentré.

1. *Smart Factory, Innovative Factory* ou « Industrie 4.0 » sont autant de manières de parler de l'usine du futur

2.4 Les tableaux de suffixes

De nombreux algorithmes de cette thèse vont reposer sur la recherche de motifs à l'intérieur d'une chaîne. Pour cette recherche, les deux méthodes les plus utilisées dans la littérature sont la création de l'arbre des suffixes de la chaîne, et la création du tableau des suffixes de la chaîne.

Les suffixes d'une chaîne, ce sont toutes les sous chaînes qui commencent à une position de la chaîne et qui se terminent à la fin de la chaîne.

L'arbre des suffixes est une structure de données arborescente qui contient tous les suffixes de la chaîne. L'arbre des suffixes est utilisé pour l'indexation de textes, la recherche de motifs et la compression de la chaîne. Ces techniques sont très utilisées dans le domaine bio-informatique.

Le tableau des suffixes contient la liste des entiers qui correspondent aux positions de début des suffixes de la chaîne, lorsque ces suffixes sont triés selon l'ordre lexicographique. La longueur du tableau des suffixes est égale à la longueur de la chaîne. La création du tableau des suffixes est complétée par celle du tableau des LCP (*Longest Common Prefix*) c'est à dire du plus long préfixe commun entre 2 entrées consécutives du tableau des suffixes. Ces deux tableaux servent, comme l'arbre des suffixes, à indexer des textes, à rechercher des motifs dans une chaîne et à compresser la chaîne.

2.4.1 L'arbre des suffixes

En 1973, Weiner [Wei73], dans « *Linear pattern matching algorithms* », est l'un des premiers à décrire et à créer l'arbre des suffixes d'un texte avec un algorithme qui est linéaire en temps, en fonction de la longueur de la chaîne.

En 1976, McCreight [McC76], publie dans l'article « *A space-economical suffix tree construction algorithm* » un nouvel algorithme de construction de l'arbre des suffixes qui, lui aussi, est linéaire en temps, et qui économise, d'après l'auteur, environ 25% d'espace par rapport à la solution de Weiner. On voit déjà que le problème de l'espace utilisé par l'arbre des suffixes est un sujet de recherche.

En 1997, Gusfield [Gus97], dans le livre « *Algorithms on strings, trees, and sequences* » donne de nombreux exemples d'utilisation de l'arbre des suffixes pour solutionner différents problèmes, en particulier en biologie comme la détection de la contamination de l'ADN (Acide DésoxyriboNucléique), ou la compression de la chaîne de lADN. On peut citer également les travaux sur le génome de Kurtz et al. (2001) [Kur+01] et ceux de Saha et al. (2008) [Sah+08]). Mais, ces types de travaux se sont heurtés assez rapidement à la taille de la cible génomique car l'utilisation des arbres des suffixes entraîne une occupation mémoire égale à environ 15 à 20 fois la taille du génome. L'utilisation du tableau des suffixes ramène cette taille à 5 fois la taille du génome (1 fois pour les données en entrée et 4 fois pour le tableau des suffixes, ceci grâce à quelques optimisations). Les travaux de Franek en 2003 [FST03], et ceux de Narisawa et al. (2007) [Nar+07]) vont dans ce sens en utilisant le tableau des suffixes en

lieu et place de l'arbre des suffixes.

Progressivement, à partir de 2003, le tableau des suffixes (« *suffix array* ») remplace l'arbre des suffixes, et les recherches se sont portées vers ce domaine.

2.4.2 Le tableau des suffixes (*suffixes array : SA)*

En 1993, Manber et Myers proposent dans « *Suffix arrays a new method for on-line string searches* » une nouvelle structure de données conceptuellement simple, qu'ils appellent tableau de suffixes, et qui remplace la structure arborescente de l'arbre des suffixes. Les auteurs indiquent que le principal avantage des tableaux de suffixes par rapport aux arbres de suffixes est qu'en pratique, les tableaux utilisent trois à cinq fois moins d'espace que les arbres. Du point de vue de la complexité, les tableaux de suffixes permettent de répondre à la question « Est-ce que W est une sous-chaîne de A ? » dans le temps en $O(P + logN)$, où P est la longueur de W, et N est la longueur de A. Ils soulignent que ce temps est compétitif avec celui de l'arbre des suffixes, et dans certains cas légèrement mieux. Le seul inconvénient, noté par les auteurs, est que les arbres de suffixes peuvent être construits en temps $O(N)$, alors que leur méthode construit les tableaux de suffixes en temps $O(NlogN)$.

Ce coefficient multiplicatif du temps en $logN$ de la solution proposé par Manber et Myers par rapport à l'arbre des suffixes sera supprimé, presque simultanément, en 2003, par les 3 travaux suivants qui reposent sur le tableau des suffixes :

- Ko et Aluru, dans « *Space efficient linear time construction of suffix arrays* » [KA03]
- Kärkkäinen et Sanders, dans « S*pace efficient linear time construction of suffix arrays* » [KS03]
- Kim, Sim, H Park et K Park, dans « *Linear-time construction of suffix arrays* » [Kim+03]

En 2007, Puglisi dans « *A taxonomy of suffix array construction algorithms* » [PST07] fait un recensement et une étude sur 14 méthodes de création des suffixes array, dont une pour l'arbre des suffixes. Et il fait un relevé très poussé des occupations mémoires et des temps de traitements. Il fournit des descriptions simples de haut niveau de ces 14 méthodes et souligne à la fois leurs différences et leurs caractéristiques communes, tout en évitant les détails de leur mise en œuvre.

Les travaux de Puglisi et al. [PSY10], et ceux plus récent de Yusufu et al. [YY15], confirment l'abandon de l'arbre des suffixes pour utiliser le tableau des suffixes, pour des raisons d'occupation mémoire. Ces algorithmes nécessitent la création du tableau des suffixes (SA), des plus longs préfixes communs (LCP) et de 2 autres tableaux (BWT,LAST). On constate, cependant, dans les résultats obtenus par Puglisi et al. [PSY10] (Tab.1), que la somme des temps pour créer les 4 tableaux préparatoires (1,912+0,17,+0,035+0,039) est égale à 144 fois le temps de l'algorithme (0,015) le plus rapide PSY1-1 pour détecter les répétitions (PSY1-4, Table.3 à droite dans la figure 2.2, page 29).

File	SA	LCP	BWT	LAST
fibo35	0.898	0.142	0.025	0.031
fibo36	0.886	0.601	0.027	0.033
fss9	0.826	0.561	0.026	0.031
fss10	0.958	0.576	0.025	0.032
periodic AVG	0.892	0.470	0.026	0.032
rand2	0.947	0.144	0.026	0.031
rand21	1.135	0.112	0.025	0.031
random AVG	1.041	0.128	0.025	0.031
ecoli	1.413	0.116	0.025	0.031
chr22	1.635	0.146	0.035	0.040
chr19	1.873	0.160	0.044	0.053
DNA AVG	1.754	0.141	0.035	0.041
prot-a	1.778	0.142	0.027	0.032
prot-b	1.971	0.159	0.034	0.039
protein AVG	1.874	0.151	0.030	0.036
bible	1.417	0.111	0.024	0.030
howto	1.912	0.178	0.035	0.039
mozilla	1.815	0.135	0.032	0.036
English AVG	1.417	0.141	0.030	0.035
AVERAGE	1.390	0.235	0.029	0.035

TABLE 2. Microseconds per letter used by each run for pr processing: SA, LCP, BWT, and LAST arrays

File	PSY1–1	PSY1–2	PSY1–3	PSY1–4	[17]
fibo35	0.054	0.052	0.019	**0.012**	0.061
fibo36	0.056	0.052	0.020	**0.012**	0.056
fss9	0.049	0.049	0.021	**0.014**	0.057
fss10	0.055	0.055	0.021	**0.013**	0.059
periodic AVG	0.054	0.052	0.020	**0.013**	0.058
rand2	**0.014**	0.016	0.017	0.017	0.052
rand21	**0.008**	0.009	0.010	0.012	0.013
random AVG	**0.011**	0.013	0.013	0.015	0.032
ecoli	**0.012**	0.014	0.015	0.015	0.031
chr22	**0.014**	0.016	0.016	0.016	0.047
chr19	**0.014**	0.020	0.022	0.016	0.051
DNA AVG	**0.013**	0.017	0.019	0.016	0.043
prot-a	**0.011**	0.013	0.013	0.013	0.028
prot-b	**0.012**	0.014	0.014	0.013	0.037
protein AVG	**0.012**	0.013	0.014	0.013	0.033
bible	0.016	0.017	0.017	**0.015**	0.049
howto	**0.015**	0.017	0.018	0.016	0.060
mozilla	**0.011**	0.013	0.013	0.013	0.032
English AVG	**0.014**	0.016	0.016	0.015	0.047
AVERAGE	0.024	0.025	0.017	**0.014**	0.045

TABLE 3. Microseconds per letter used by each run for PSY1 and the algorithm of [17]

FIGURE 2.2 – Résultats de Puglisi pour la recherche des répétitions dans une chaîne extrait de « *Fast, Practical Algorithms for Computing All the Repeats in a String* », Simon J. Puglisi, W. F. Smyth and Munina Yusufu

Kärkkäinen et Kempa dans « *Faster external memory LCP array construction* » [KK16] considèrent que le tableau des suffixes associé au tableau LCP sont les 2 structures les plus importantes pour indexer ou compresser les index des textes car elle permettent de créer ces tableaux sur un support extérieur à la mémoire pour traiter les très grandes chaînes comme les génomes : « Le tableau des suffixes (liste des suffixes d'un texte triée par ordre lexicographique) est la structure de données la plus importante du traitement moderne des chaînes. Il est fréquemment complété par le tableau LCP (le tableau des préfixes communs les plus Longs), qui stocke les longueurs des préfixes communs les plus longs entre deux suffixes lexicographiquement adjacents. Ensemble, ils constituent la base de puissants index de texte tels que les tableaux de suffixes améliorés et de nombreux index compressés en texte intégral. ». C'est également notre avis.

Plus récemment, les travaux de Louza et al. (2017) [Lou+17] utilisent eGSA (*External memory generalized suffix and LCP arrays construction*) qui repose sur le tableau des suffixes et des LCP pour extraire les répétitions dans une chaîne. Ils fusionnent la création de ces 2 tableaux dans leur algorithme et ils les stockent, lors de leur création, sur des mémoires externes ; ils considèrent que c'est le premier algorithme qui construit à la fois le tableau des suffixes et le tableau LCP sur des mémoires externes dans le même algorithme. Cette création simultanée des 2 tableaux est un point important que nous utiliserons pour la construction de notre méthode (voir chapitre 3.6).

Nous montrerons qu'il existe une alternative (chapitre 3.6, page 57) à la création de ces tableaux qui sont très coûteux en temps. Cette alternative intègre le calcul des répétitions dans un algorithme de tri des suffixes d'une chaîne. Il tient compte du nombre de répétitions et de la longueur de la répétition que

l'on souhaite trouver. Cette alternative ne nécessite pas de trier la totalité des suffixes contrairement à toutes les solutions que nous avons identifiées dans cet état de l'art. C'est cette méthode que nous développons dans la méthode SAP (pour *Suffixe Array for Pattern*). L'algorithme général (algorithme.2) de cette méthode est en page 59. L'algorithme optimisé (algorithme.3) est décrit dans l'annexe 3.6.3 (page 61).

2.5 Les ontologies

Dans ces travaux de recherche nous Utilisons souvent des ontologies comme un système de représentation des connaissances. C'est un outil qui nous permet de modéliser les connaissances du domaine. Modèle que l'utilisateur modifie et adapte à ses besoins si necessaire. Pour créer les ontologies, nous nous sommes inspirés du logiciel « Protégé » de l'Université de Stanford. « Protégé » est le logiciel le plus populaire pour la création des ontologies et des connaissances ceci d'après [MMS+11] qui reprend les chiffres de l'étude de 2007 de Jorge Cardoso [Car07]. L'utilisation de « Protégé » est très conviviale et très simple. Nous avons pris le même type d'ergonomie que celle du logiciel « Protégé ». Pour développer les ontologies nous avons utilisé une démarche déjà validée dans d'autres domaines et connue sous le nom de « éthontology » [Cor+05]

Depuis le début, les ontologies ont été considérées comme un élément permettant à la méthode KDD d'être plus précise et plus efficace. Usama Fayyad et les co-auteurs le précisent bien dans le paragraphe « *Concluding Remarks : The Potential Role of AI in KDD* » que « *Knowledge representation includes ontologies, new concepts for representing, storing, and accessing knowledge. Also included are schemes for representing knowledge and allowing the use of prior human knowledge about the underlying process by the KDD system* » [FPS96a]. L'article "*Semantic Data Mining : A Survey of Ontology-based Approaches*" fait une synthèse assez complète du rôle des ontologies dans le datamining [DWL15].

2.6 Les outils sur le marché

Nous avons fait des tests avec deux produits du marché qui permettent de lire et d'interpréter des fichiers PDF. Le premier produit est Word™ de Microsoft™ qui permet de lire directement les fichiers PDF, le deuxième est Acrobate DC™ d'Adobe™ qui permet de créer des fichiers *.docx qui sont relus par Word. En entrée, nous utilisons le fichier PDF, de 36 pages, accessible à l'adresse http ://www.1a3i/~r3d/data/ [2].

Les résultats de la lecture avec Word™ de Microsoft™ (Microsoft Office Professionnelle™ 2013 version 15.0.5137.1000) sont très décevants. Sur les 36 folios du dossier seulement 2 folios sont reconnaissables quand ils s'affichent dans

2. l'ensemble des fichiers utilisé dans le cadre de cette thèse sont disponibles à l'adresse : http ://www.1a3i/~r3d/data/.

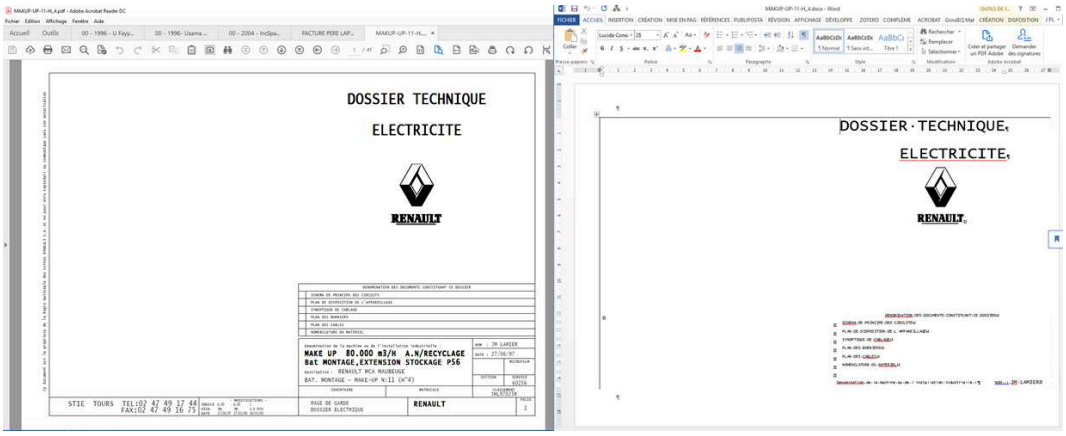


FIGURE 2.3 – Comparaison de la page de garde en PDF (à gauche) et dans Word™ (à droite)

Word™, ce sont les folios de la page de garde (folio 1) et des moteurs (folio 4). Les 34 autres sont illisibles. Même quand Word™ arrive à afficher le folio (uniquement pour ces 2 folios), la qualité de l'information affichée par rapport à l'original en PDF est médiocre : voir la figure 2.3, page 31 pour la page de garde et la figure 2.4, page 32 pour le folio des moteurs.

Les résultats avec Acrobat DC™ d'Adobe™ (version 2019.012.20040) sont meilleurs. 30 folios sont bien lus sur les 36 folios du dossier. Cinq folios subissent une rotation à 90° : c'est le cas du folio des moteurs (folio 4). D'autres présentent des défauts d'affichage : par exemple, le folio 4 des moteurs dans la figure 2.4 est comparé à l'original et au résultat de Word™. On constate de nombreux défauts d'affichage des graphismes des symboles pour Adobe™ et un déplacement important des moteurs pour Microsoft™.

En conclusion, Abobe™ et Microsoft™ lisent les fichiers PDF sans les restructurer.

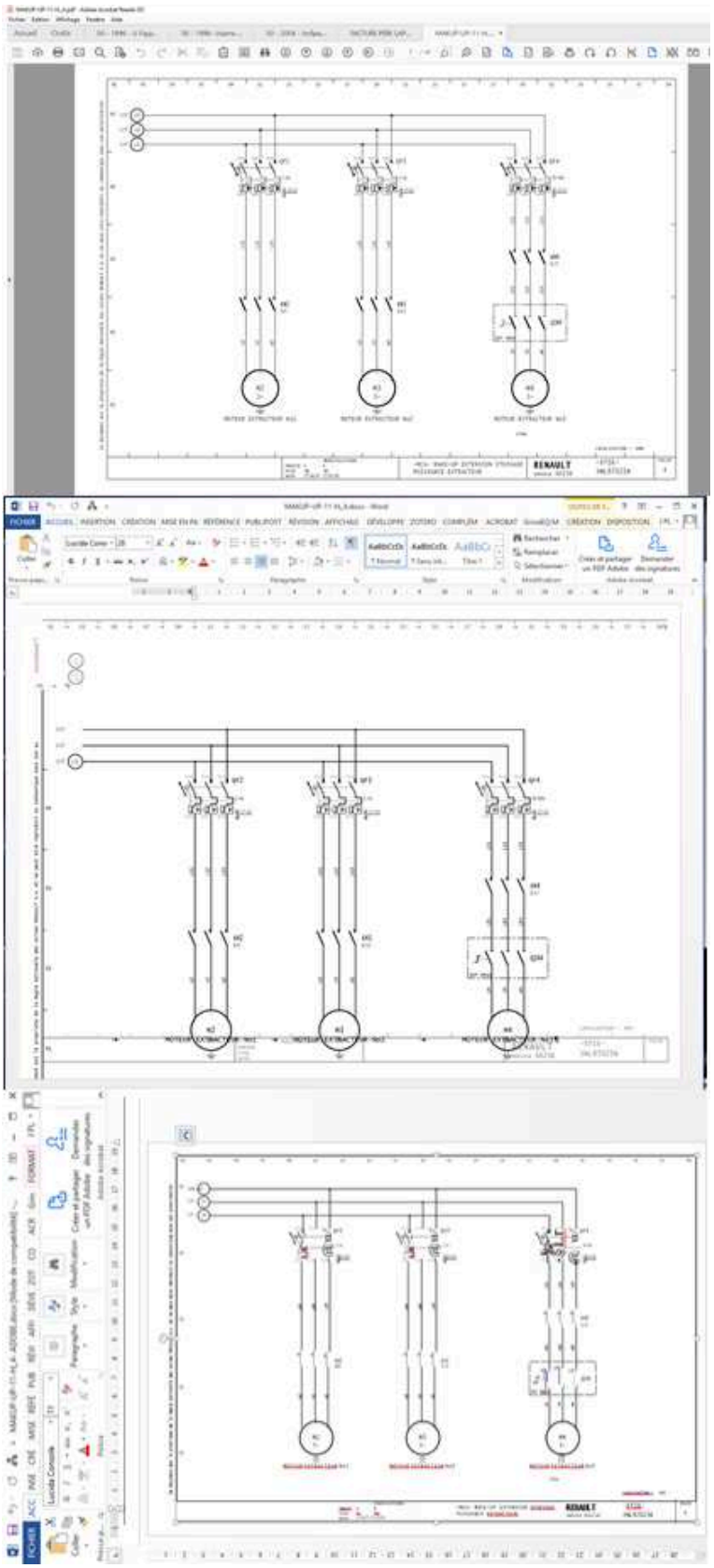


FIGURE 2.4 – Comparaison du folio 4 des moteurs en PDF (en haut) avec Microsoft™ (au centre) et Adobe™ (en bas après rotation)

En figure 2.5, page 33, nous présentons le folio 4 des moteurs produit par la méthode (A)KDD, avant la restructuration complète du dossier, et, exporté dans Word™ (la même version de Word™ que celle utilisée pour lire le fichier PDF). On peut constater que le folio généré est identique au folio stocké dans le fichier PDF, voir l'image (du haut) figure 2.4, page 32.

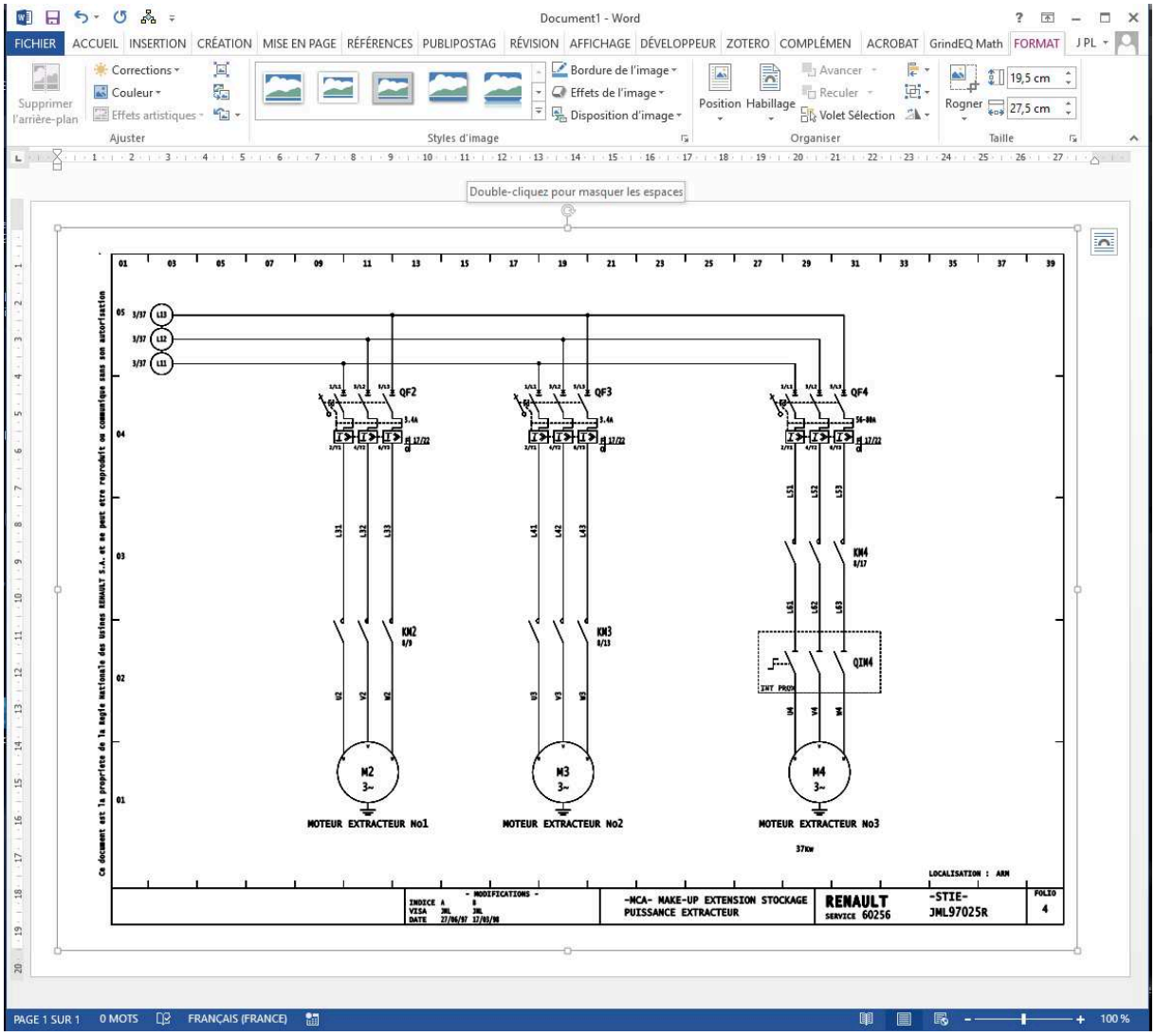


FIGURE 2.5 – Le folio 4 des moteurs repris par la méthode (A)KDD avant restructuration

Section 3

La méthode (A)KDD

En 1996, dans « *The KDD process for extracting useful knowledge from volumes of data* » [FPS96b], Usama Fayyad donne la définition suivante de la méthode KDD : « *The nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data* », que l'on peut traduire par : le processus non trivial d'identification de motifs valides, nouveaux, potentiellement utiles, et finalement compréhensibles des données.

Pour Usama Fayyad, chaque mot de cette définition est important. Dans l'article, il les reprend un par un pour dire pourquoi chacun de ces mots est fondamental dans la méthode KDD. Nous retiendrons 2 mots qu'il definit : a) la méthode KDD est un « *Process* » : c'est un processus car la méthode nécessite plusieurs phases qui peuvent être toutes répétées de façon itératives « *implies there are many steps involving data preparation, search for patterns, knowledge evaluation, and refinement—all repeated in multiple iterations* », b) le mot « *nontrivial* », car il considère que la méthode n'est pas un simple calcul de quantité mais la recherche de structure complexe (« *process is assumed to be nontrivial in that it goes beyond computing closed-form quantities ; that is, it must involve search for structure, models, patterns, or parameters.* »).

Dans la figure 2.1 (page 25), il propose de décomposer la méthode en cinq phases principales. Nous appliquons ces phases à la restructuration des documents graphiques. Nous regroupons la phase 1 sélection et la phase 2 préprocessing. Car les deux, dans notre cas d'usage, sont des phases de préparation des données. Dans la méthode (A)KDD la phase 1 est la phase de « Sélection, nettoyage des données ».

3.1 Passage de KDD à (A)KDD

3.1.1 Phase 1 : La sélection et le nettoyage

Cette phase comprend les opérations de base, telles que :

— Le choix des données à traiter : dans notre cas d'usage, cela revient à choisir les dossiers, les folios (si nécessaire) et parties de folios (si nécessaire) qui sont à restructurer.

— Le choix de l'environnement : dans notre cas d'usage, c'est le choix des bases graphiques et des bases de connaissances, ainsi que l'ontologie qui sert tout au long de la restructuration.
— La suppression du bruit dans le fichier d'entrée : cette fonctionnalité s'applique parfaitement aux données graphiques que nous traitons.
— La suppression des valeurs aberrantes : c'est un sujet que nous ne traitons pas directement. Nous partons du principe que les données en entrée sont justes dans la majorité des cas (elles sont issues de logiciels de CAO). Et, si ce n'est pas le cas, c'est l'utilisateur qui détermine la vérité par les choix qu'il fait tout au long du processus.
— La suppression des doublons pour une meilleure analyse : c'est un de nos problèmes de nettoyage des données (Voir paragraphe 3.4.2).

3.1.2 Phase 2 : La Codification

Usama Fayyad indique que cette phase comprend la recherche des fonctionnalités utiles pour représenter les données en utilisant la réduction de dimensionnalité et trouver des représentations invariantes dans les données.

Pour la restructuration des documents déstructurés, nous faisons la réduction de la dimensionnalité en transformant l'espace 2D des données graphiques contenues dans le fichier PDF, en un espace 1D matérialisé par une chaîne de codes.

Dans l'ontologie [1] que nous développons, un objet peut avoir plusieurs instances. Par exemple, la notion de moteur peut être associée à plusieurs objets (les moteurs monophasés, les moteurs triphasés, etc ...) et chaque objet peut avoir plusieurs instances (plusieurs représentations différentes). Pour trouver les représentations invariantes dans les données, nous partons de l'hypothèse que les logiciels de DAO/CAO ou de PAO dessinent les objets toujours de la même façon. Si le même objet a deux ou plusieurs représentations différentes, cela ne pose pas de problème à la solution que nous développons, puisque dans l'ontologie on peut associer plusieurs instances à un objet.

Les travaux cité au chapitres sur l'état de l'art (voir chapitre 2), que nous avons étudiés, sur la restructuration des fichiers PDF, ne tiennent jamais compte de cette possibilité d'avoir les objets graphiques toujours dans le même ordre chronologique. Ils prennent d'ailleurs des exemples qui indiquent que dans certains cas les objets, en particulier les textes, ne suivent pas toujours l'ordre chronologique d'apparition. Par exemple, dans les unes des revues, des lettres ou des blocs de lettres ne sont pas rangés chronologiquement. Ils sont donc dans l'impossibilité d'utiliser cet ordre chronologique des objets (voir [BI], [Rob01]). Nous pensons que, d'une part, ce cas est très rare, même impossible en CAO (nous ne l'avons jamais rencontré dans les fichiers issus des logiciels de CAO), et que, d'autre part, ce désordre chronologique peut aussi provenir du fait qu'ils utilisent des outils (qu'ils n'ont pas développés) pour extraire les textes. Il est fort possible que ces outils d'extraction ne respectent pas l'ordre des textes dans

1. Les ontologies sont utilisées dans la méthode (A)KDD comme un modèle de données représentatif des concepts du domaine, de la façon la plus classique qu'il soit.

les fichiers PDF, car ils font des associations de lettres pour former des mots. Ces associations ne sont pas toujours correctes (c'est le constat de [BBH03], [LB95], [SB95]), et de ce fait, ces associations incorrectes peuvent entraîner, une restitution de l'information, qui ne respecte pas l'ordre chronologique d'origine.

Dans notre cas d'usage, nous n'avons pas trouvé ce type d'anomalie dans les fichiers que nous traitons [2].

Deux raisons principales peuvent expliquer cette vision différente de la chronologie des objets, entre les travaux de certains chercheurs et les nôtres. La première raison, c'est que les logiciels de CAO (ou de DAO et PAO) obligent les utilisateurs à respecter de nombreuses règles afin que les traitements de CAO puissent s'effectuer correctement (par exemple : si un moteur a pour nom « MT34 », il est impossible de le nommer en créant 2 blocs de texte, l'un avec « MT » et l'autre avec « 34 »). La deuxième raison, c'est que nous avons développé notre propre extracteur d'informations des fichiers PDF (et des autres fichiers vectoriels). Cet extracteur respecte parfaitement l'ordre chronologique d'apparition des objets graphiques dans le fichier PDF, et il ne fait aucune association, ce qui n'est pas le cas de certains extracteurs (comme nous venons de le voir). Cet extracteur n'est pas décrit dans ces travaux car c'est un pur travail d'ingénierie qui a été réalisé avant la thèse.

> Tous nos travaux reposent sur l'hypothèse que l'ordre d'apparition des objets graphiques d'un même objet est toujours le même. S'il est différent, c'est qu'il correspond à deux instances différentes du même objet.

Ceci nous permet de définir des fonctionnalités (comme le mentionne Usama Fayyad) utiles pour trouver des représentations invariantes des données. Fonctionnalités qui s'adaptent chaque fois à l'objet ou aux objets graphiques que l'on recherche et ceci en tenant compte de l'ordre chronologique.

Il existe deux avantages à définir des fonctionnalités et des algorithmes qui prennent en compte l'ordre chronologique. Tout d'abord la rapidité de traitement des algorithmes, en effet, il est plus rapide de comparer un objet avec son précédent que de faire une recherche sur l'ensemble des objets. Ensuite la possibilité de transformer l'espace 2D en un espace 1D et de transformer la recherche d'un graphisme en 2D en recherche d'une sous chaîne dans une chaîne.

3.1.3 Phase 3 : La fouille de données

La fouille de données est intimement liée à la codification. Pour la fouille de données, Usama Fayyad précise trois points importants pour la méthode KDD, il faut :

— choisir la fonction de fouille de donnés : ce qui comprend le choix de l'objectif du motif attendu par l'algorithme d'exploration de données (par exemple, résumé, classification, régression et regroupement). Dans

2. Les fichiers de test sont disponibles à l'adresse http ://www.1a3i/~r3d/data/

notre cas d'usage, le choix du motif est différent en fonction de l'objet graphique que l'on recherche (comme pour la codification qui est étroitement liée au motif). Nous expliquons le choix du motif dans chacune des tâches de codification des étapes incrémentales de restructuration (voir chapitres 4.1 à 4.3.3).

— choisir le ou les algorithmes d'exploration de données : c'est à dire la ou les méthodes de recherche à utiliser pour trouver des motifs (*patterns*) dans les données, avec le choix des paramètres appropriés. Dans notre cas d'usage ils dérivent de la méthode « SAP » (voir chapitre 3.6, page 57).

— exploration de données : comprend la recherche de motifs dans les données d'entrée en appliquant la ou les algorithmes d'exploration de données choisies, pour en extraire tous les motifs.

3.1.4 Phase 4 : L'interprétation et l'évaluation

Pour Usama Fayyad, cette phase comprend l'interprétation des motifs trouvés et permet éventuellement de revenir à l'une des phases précédentes. Dans notre cas d'usage, l'interprétation des motifs n'est faite que par l'utilisateur. L'ordinateur propose les résultats de la recherche des motifs et c'est l'utilisateur qui les valide ou les corrige. En cas de correction, il est important, comme le souligne Usama Fayyad de pouvoir revenir à l'une des étapes précédentes, pour refaire les traitements. Il est également très important de pouvoir réitérer le traitement de la phase en modifiant certains paramètres.

Cette phase comprend donc, la visualisation des motifs extraits, la présentation sous une forme compréhensible par les utilisateurs des motifs retenus, et, la possibilité de modifier, de rajouter ou de supprimer les éléments sélectionnés par l'ordinateur. C'est dans cette phase que les connaissances acquises sont incluses dans un système de performance. Dans notre cas d'usage, le système performant est matérialisé par une base de connaissances pilotée par l'ontologie et une base graphique des motifs. C'est exactement ce que nous faisons dans la méthode (A)KDD (voir figure 2.1 page 25).

3.1.5 La nécessité d'une méthode

Usama Fayyad place la méthode KDD à l'intersection de la recherche dans des domaines tels que les bases de données, l'apprentissage automatique, la reconnaissance de formes, la statistique, l'intelligence artificielle, le raisonnement avec incertitude, l'acquisition de connaissances, la visualisation de données, la découverte automatique, la recherche d'informations. D'après lui, les logiciels, reposant sur la méthode KDD incorporent des théories, des algorithmes et des méthodes issus de tous ces domaines (voir « *From Data Mining to Knowledge Discovery in Databases* », paragraphe « *The Interdisciplinary Nature of KDD* »)[FPS96a].

C'est exactement la vision que décrivent les auteurs dans l'article « *Visual Analytics : Definition, Process and Challenges* »[Kei+08]. Vision qu'ils mo-

délisent dans le schéma ci-dessous (voir figure 3.1). Les auteurs considèrent que « l'acquisition de données brutes n'est plus le problème : c'est la capacité à identifier des méthodes et des modèles qui peuvent transformer les données en connaissances fiables et prouvables ». C'est exactement l'objectif de nos travaux : développer une méthode qui intègre toutes ces disciplines pour restructurer les documents déstructurés de type graphique. Cet objectif se matérialise par notre méthode (A)KDD.

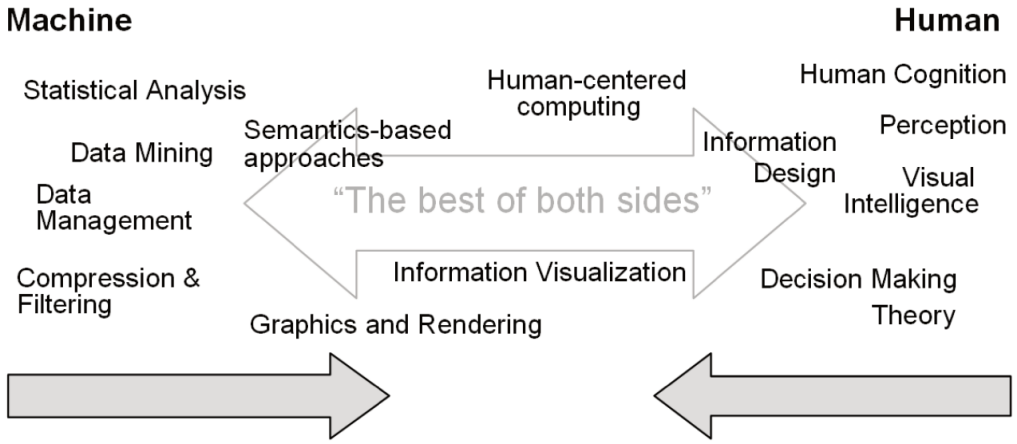


FIGURE 3.1 – Les disciplines scientifiques pour améliorer la répartition du travail entre l'homme et la machine : extrait de « *Visual Analytics : Definition, Process and Challenge* » Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, Guy Melançon [Kei+08]

Dans le cas de la restructuration des documents déstructurés graphiques, nous ajoutons, à l'ensemble des domaines cités, les ontologies, l'analyse des chaînes (*stringology*), la maîtrise de la DAO, la maîtrise de la CAO et la maîtrise de la PAO.

Dans « Visualisation d'information » [MEL15], Guy Melaçon, précise que : « Le succès de la fouille de ces données tient à la découverte de motifs structuraux et à leur interprétation, qui reste l'affaire d'un utilisateur humain. Lui seul peut juger de la pertinence d'un résultat, en apprécier le sens et l'impact potentiel, et le cas échéant prendre les bonnes décisions ». C'est le « (A) » de l'aspect Antropocentré de la méthode (A)KDD .

Usama Fayyad indique que « *The KDD process is interactive and iterative (with many decisions made by the user), involving numerous step* » (voir « *From Data Mining to Knowledge Discovery in Databases*»[FPS96a], paragraphe « *The KDD process* »). C'est le côté Itératif, Interactif et Impliquant de la méthode KDD (les trois « I » du KDD).

A ces trois « I » de la méthode KDD, la méthode (A)KDD ajoute le « I » de Incremental et le « A » de Anthropocentré ce qui nous donne l'équation : $IA + 3I = (A)KDD$ ($IA + 3I$, presque l'anagramme de 1A3i).

3.2 Aspect incrémental de la restructuration de la méthode (A)KDD

La méthode (A)KDD reprend tous les aspects de la méthode KDD, comme nous venons de le voir, et ajoute les ontologies, comme le propose Usama Fayyad en 1996 dans « *From Data Mining to Knowledge Discovery in Databases* », l'aspect incrémental et l'aspect Antropocentré. Les ontologies sont incluses dans la base de connaissances de la méthode (A)KDD.

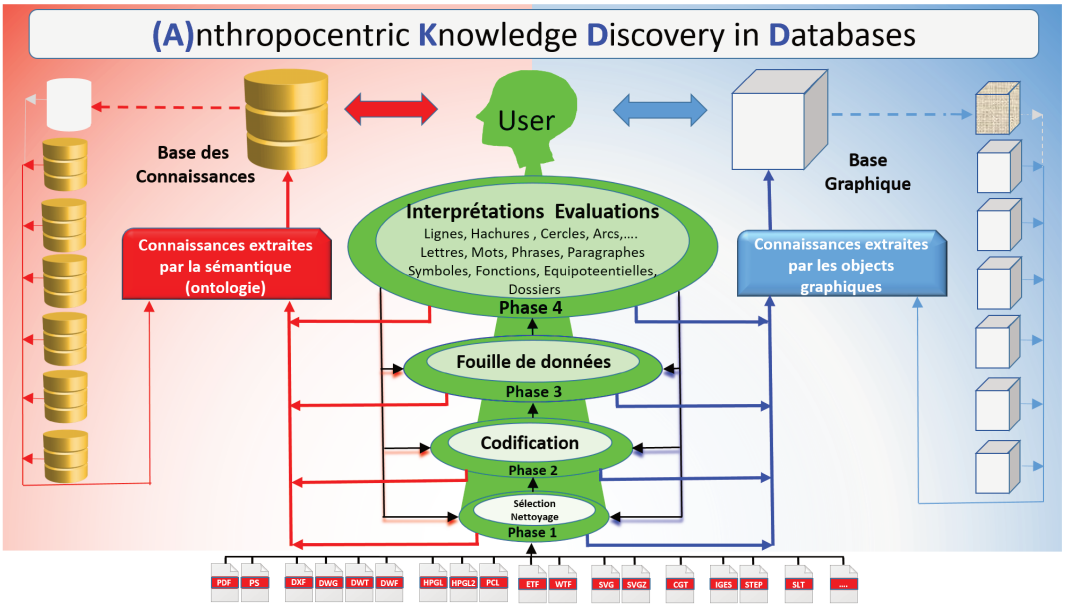


FIGURE 3.2 – Les quatre phases de la méthode (A)KDD

La méthode (A)KDD repose sur (voir figure 3.2) :

— une base de connaissances, où les ontologies ont un grand rôle. Elles contiennent la définition de tous les objets qui sont manipulés et de toutes les instances associées à ces objets. Les ontologies enregistrent la structure de tous les objets du dossier après restructuration. Les ontologies permettent de définir les types de connexions, pour la schématique électrique ce sont des équipotentielles (de puissances, de télécommandes). Elles définissent aussi toutes les classes de symboles, les symboles de présentation comme les cartouches, les tableaux de fils ou les symboles électriques en les classant par type (sectionneurs, protections, signalisation, etc...). C'est dans l'ontologie que l'on trouve tous les attributs des symboles avec leur définition.

— une base graphique qui contient le graphisme de toutes les instances des objets de la base de connaissances et qui contient également le résultat de la restructuration des documents en entrée. Cette base graphique est enregistrée au format DXF et DWG. C'est celui qui permet de mieux tenir compte de la restructuration que nous réalisons. C'est un format que tous les logiciels (ou presque) de DAO et CAO savent lire.

— un processus incrémental de la reconstruction, on commence à reconstruire des objets basiques, puis petit à petit des objets plus structurés

pour terminer par la structure complète du dossier (voir figure 3.3, page 40).

— une implication à tous les niveaux de l'utilisateur. Dans la méthode KDD l'utilisateur intervient essentiellement pour déclencher une itération du processus. Dans notre méthode (A)KDD, c'est l'utilisateur qui passe d'une étape à l'autre. C'est l'utilisateur qui déclenche chaque phase. C'est l'utilisateur qui valide les propositions des algorithmes. C'est l'utilisateur qui déclenche l'enregistrement dans la base de connaissances. C'est l'utilisateur qui décide d'itérer une ou plusieurs phases. C'est l'utilisateur qui pilote tout le processus, pas la machine.

Pour restructurer un document PDF, il faut procéder étape par étape et faire le chemin inverse de la déstructuration qui a produit le fichier PDF (processus de déstructuration que nous avons décrit dans l'annexe A).

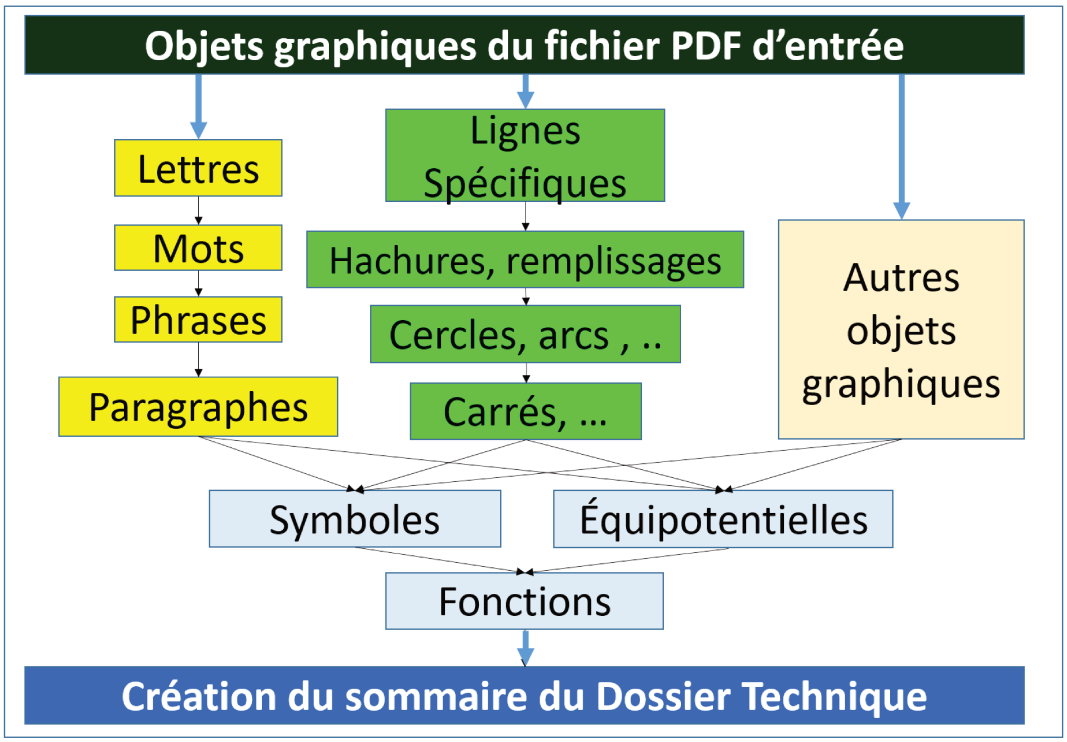


FIGURE 3.3 – Les flux des traitements du processus incrémental de la méthode (A)KDD

Pour chacune des étapes incrémentales que nous allons décrire, les 3 dernières phases de la méthode (A)KDD seront appliquées : la phase 2 de codification, la phase 3 de fouille de données et la phase 4 d'interprétation et d'évaluation. La première phase de sélection et de nettoyage est faite une fois pour l'ensemble du fichier PDF en entrée, elle n'est pas répétée à chaque étape incrémentale.

Les étapes de cette restructuration incrémentale sont les suivantes :

— restructurer les objets graphiques en commençant par :
  — les traits discontinus : les traits mixtes, les traits d'axe, les traits pointillés,
  — les remplissages et les hachurages,

  - les formes coniques : les cercles, les ellipses, les arcs de cercle, les arcs d'ellipse,
  - les triangles, les carrés, les rectangles, les parallélogrammes,
- restructurer les objets textuels en commençant par :
  - les caractères (si nécessaire [3]),
  - les mots,
  - les phrases,
  - les paragraphes (si nécessaire [4]),
- restructurer les symboles et les liaisons,
- restructurer les fonctions,
- restructurer le dossier.

Un objet graphique vectoriel est un objet composé de 1 à N segments de droite. Un objet graphique textuel est un objet composé de 1 à N lettres, nous verrons paragraphe 4.2.3 que les lettres peuvent-être des objets graphiques vectoriels.

L'objectif premier de cette restructuration incrémentale est de créer des objets graphiques de « plus haut niveau » qui remplacent (en lieu et place) les objets qui participent à leur création. De cette façon, à chaque incrémentation le nombre d'objets graphiques en entrée des algorithmes diminue. Ce point est très important car l'utilisateur est présent à chaque étape et il ne faut pas que les temps de traitement, et donc d'attente, soient trop importants.

Diminuer le nombre d'objets en entrée de chaque étape incrémentale, et donc de chaque algorithme de traitement, est l'une des principales préoccupations de la méthode (A)KDD.

3.2.1 La restructuration des objets graphiques vectoriels.

Un objet graphique ou un élément graphique [5] vectoriel est :
- soit un objet graphique vectoriel de base, c'est à dire un segment de droite,
- soit l'un des objets obtenus par les restructurations précédentes ou le cumul de plusieurs de ces restructurations (exemple : une ellipse hachurée avec des traits interrompus).

Si l'on développe les opérations de restructuration de l'exemple de l'ellipse hachurée de traits interrompus il faut :
- trouver le contour de l'ellipse. Dans le fichier PDF, il n'existe pas en tant que tel. Le contour est composé de segments de droite qu'il faut ré-assembler. L'ensemble de ces segments de droite forme l'ellipse (entre 150 et 600 segments de droite).
- puis, trouver toutes les lignes en traits interrompus. La ligne n'existe pas en tant que telle dans le fichier PDF. Chaque ligne est composée d'une

3. Les caractères dans le fichier PDF peuvent ne pas être déstructurés (voir 4.2.2).
4. Les paragraphes sont peu utilisés en schématique électrique (voir 4.2.5).
5. Les 2 termes « objet » et « élément » représentent la même notion, l'objet est associé à une notion plus structurante.

succession de segments de droite qu'il faut ré-assembler. L'ensemble des segments forme la ligne en traits discontinus (entre 10 et 200 segments, suivant la longueur de la ligne).

— ensuite, trouver l'ensemble des lignes qui composent le hachurage. Nous avons trouvé dans l'étape précédente les lignes en traits pointillés, maintenant il faut les assembler pour former le hachurage.

— et enfin, dernière opération, on vérifie que le hachurage correspond bien à l'ellipse que nous avons trouvée à la première étape, et on l'associe à l'ellipse. De cette façon, nous n'avons plus qu'un objet graphique (une ellipse hachurée) qui remplace tous ces segments de droite (entre mille et deux mille).

Les objets graphiques vectoriels doivent être restructurés dans un ordre précis.

Premièrement, les segments de droite en traits discontinus.

Comme nous l'avons vu dans le préambule, pour des raisons de correspondance entre ce qui est vu à l'écran et ce qui est imprimé, les segments de droite qui matérialisent les types de traits autres que continus, sont décomposés en une multitude de petits segments de droite. C'est le cas pour : le trait interrompu, le trait mixte, le trait mixte à 2 tirets, le trait pointillé, le trait double pointillé.

La matérialisation (taille, longueur des traits, espacement des traits) de tous ces types de traits est définie par la norme (NF E04-520), mais elle n'est jamais respectée en totalité par les logiciels de DAO, CAO et PAO.

Ci-dessous, dans la figure 3.4, les trois types de traits les plus présents en schématique sont dessinés.

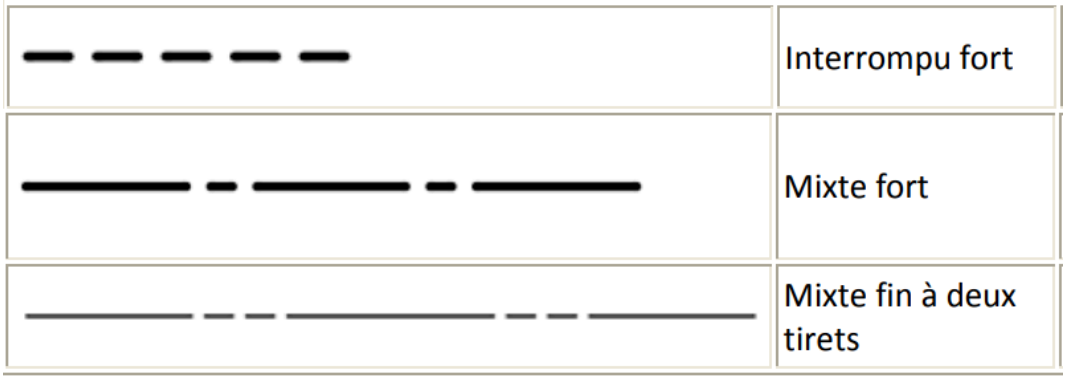


FIGURE 3.4 – Exemples de types de traits

La restructuration des segments de droite consiste à rassembler tous les segments de droite qui appartiennent à la même ligne et qui matérialisent une ligne en traits discontinus (voir paragraphe 4.1.1, page 67). C'est le premier objet graphique à restructurer, car il peut rentrer dans la composition d'objets graphiques comme les ellipses, les hachurages, etc.

Prenons un exemple. Un carré de 200 millimètres de côté tracé avec un trait interrompu. Ce carré est composé d'environ 160 petits segments de 3 millimètres

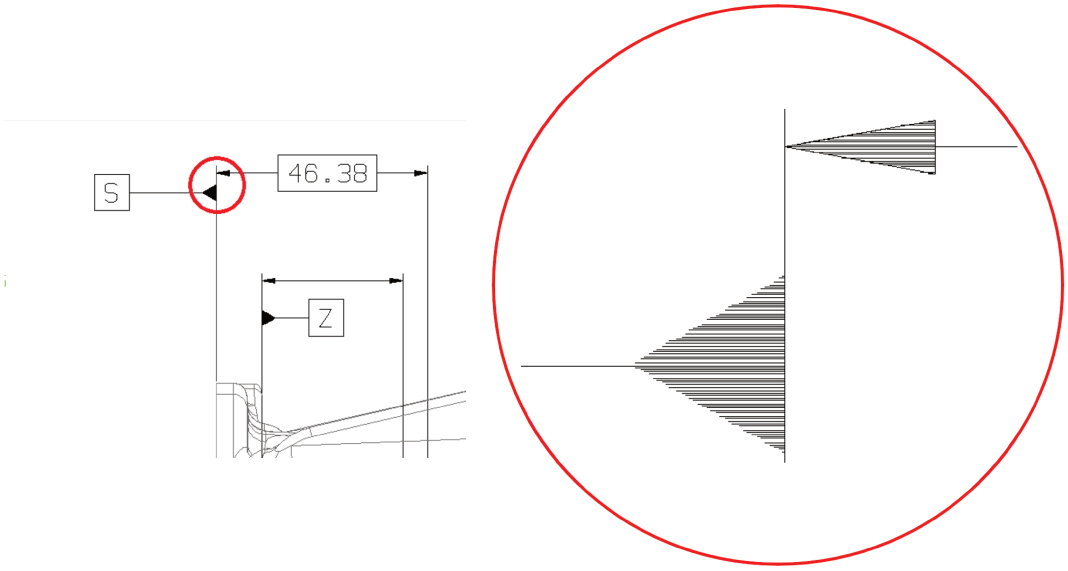


FIGURE 3.5 – Exemples de hachurages d'une flèche

et d'autant d'espaces blancs. Dans cette phase de restructuration des lignes discontinues, ces 160 petits segments de droite seront remplacés par 4 segments de droite : on voit immédiatement le gain. On divise par 40 le nombre d'objets pour matérialiser la même information.

Deuxièmement, les remplissages et les hachures.

Pour remplir ou hachurer une forme, on utilise des segments parallèles (voir figure 3.5). Par exemple, sur une table traçante, si l'on utilise une plume de diamètre 0,1 millimètre (c'est la plume que l'on utilise habituellement pour des raisons esthétiques de rendu de l'aplat) il faut tracer des segments parallèles tous les 0,05 millimètres pour que le remplissage soit correct. Si l'on prend un carré de 10 millimètres de côté, il faudra 200 segments pour le remplir.

Dans l'exemple du logo de Renault (voir figure 3.6, page 44), le nombre de segments de droite utilisés pour le remplissage est de 5 334 segments. Dans le cas des flèches, c'est environ 70 pour la flèche la plus grosse et une vingtaine pour la flèche fine.

Rappel : avant de rechercher les hachures, il faut d'abord restructurer les segments de droite en trait interrompus, car les hachures peuvent être dessinées dans ce type de trait.

L'étape incrémentale de restructuration des hachures consiste à assembler tous les segments de droite qui représentent ces hachures ou remplissages pour les remplacer par l'objet de plus haut niveau qui matérialise ces hachures (voir paragraphe 4.1.2, page 74).

Troisièmement, les coniques et les parties de coniques (cercles, ellipses, arcs de cercle, arcs d'ellipse).

Si l'on prend, par exemple, le cas des cercles dans le fichier PDF, ils sont composés de 360 petits segments qui correspondent chacun à la corde d'un angle d'un degré. Ce nombre de 360 n'est pas fixe, il est fonction du rayon. C'est, en général, le nombre de segments que l'on utilise pour un cercle de 25 millimètres

de rayon. Ce nombre augmente quand le rayon du cercle augmente (en général de telle sorte que la longueur du segment soit d'environ 0.5 millimètre) et diminue quand le rayon diminue.

L'étape incrémentale de restructuration des cercles, arcs de cercles, ellipses, arcs d'ellipses, consiste à assembler tous les segments de droite qui représentent ces coniques pour les remplacer, en lieu et place, par un objet de plus haut niveau qui matérialise cette conique (voir paragraphe 4.1.3, page 76).

Dans le cas du cercle de 25 millimètres, on divise par 360 le nombre d'objets nécessaires pour matérialiser la même information.

Quatrièmement, les formes géométriques simples, comme les carrés, les rectangles, les triangles, les losanges, les parallélogrammes.

L'étape incrémentale de restructuration des formes simples consiste à assembler tous les segments de droite qui représentent ces formes simples pour les remplacer, en lieu et place, par l'objet de plus haut niveau qui matérialise ces formes simples (voir paragraphe 4.1.4, page 78).

La restructuration de ces formes ne diminue que de très peu le nombre des objets en entrée (comparé aux 3 premières restructurations). C'est pour cette raison qu'il n'est pas toujours nécessaire de la faire. Par contre, la restructuration de ces objets peut apporter des objets différenciants de plus haut niveau, qui, dans certains types de schémas, en particulier les schémas de bâtiment, sont très intéressants.

En conclusion de la restructuration des objets graphiques

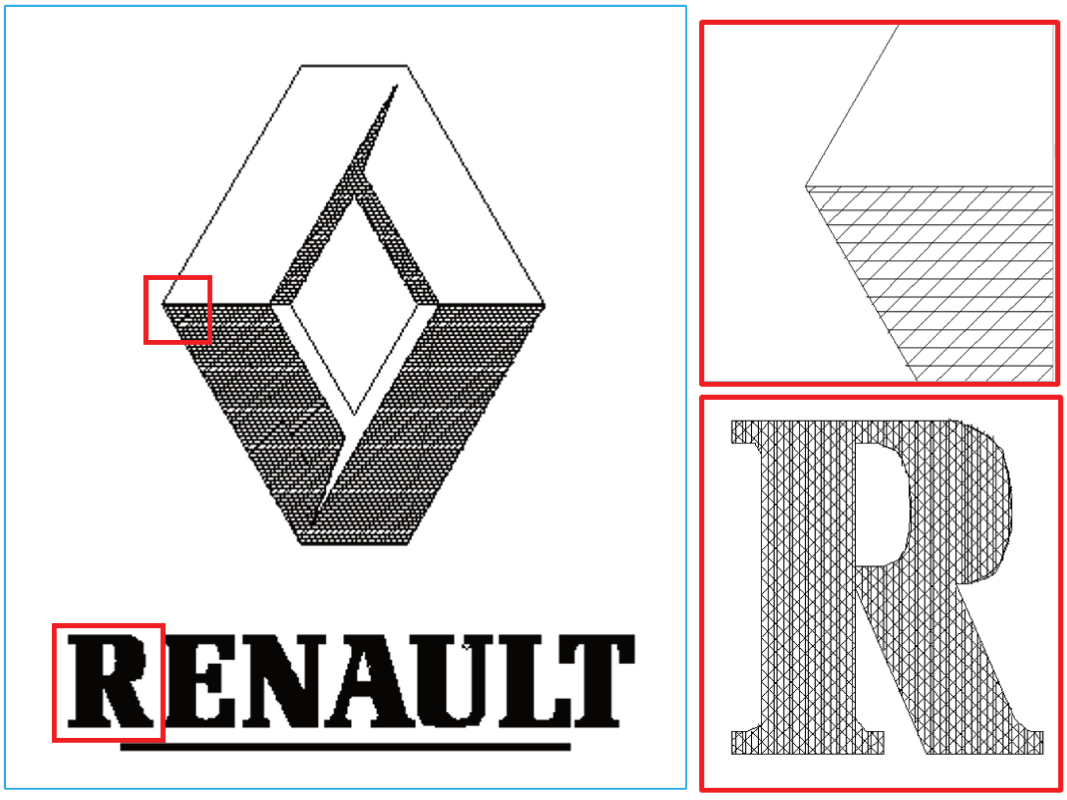


FIGURE 3.6 – Exemples de hachurages d'un logo

Il est important de noter que ces restructurations diminuent de façon très importante le nombre d'objets du fichier d'entrée. C'est un point majeur de la méthode développée. Pour rappel, l'objectif est de manipuler, à chaque étape incrémentale, de moins en moins d'objets graphiques ou textuels (voir le paragraphe 3.2.2 ci-dessous). Le seul but est que le temps de traitement soit le plus court possible pour être acceptable par l'utilisateur, qui est présent pendant tout le processus de restructuration.

3.2.2 La restructuration des objets graphiques textuels

Un objet textuel, ou un élément textuel, est une lettre, un mot, une phrase ou un paragraphe.

Une lettre, ou un caractère, peut avoir deux représentations très différentes :
— soit le caractère est enregistré dans le fichier d'entrée sous forme d'un glyphe c'est à dire sous la forme d'une ou plusieurs polylignes remplies
— soit c'est une lettre, au sens informatique du terme, c'est à dire un code qui indique le caractère dans un alphabet et une police avec les indications de nom de la police de caractère, de taille, de style, etc.

Un mot est l'association de plusieurs lettres qui forment un tout. Un mot peut être horizontal, vertical ou écrit avec un angle quelconque. Un mot peut être composé d'une seule lettre.

Une phrase est l'association d'un à plusieurs mots qui ne se terminent pas forcément par un point mais qui forment un tout. Une phrase est sur une seule ligne (verticale, horizontale ou oblique). Cette définition est différente de celle utilisée dans les traitements de textes. Pour un traitement de texte, une phrase peut-être sur plusieurs lignes et elle se termine par un point (normalement).

Un paragraphe est l'association d'une ou plusieurs phrases qui forment un tout. La notion de paragraphe est très peu présente en schématique, mais elle peut exister dans de la DAO technique.

Pour restructurer les textes nous avons quatre étapes incrémentales qui s'enchaînent les unes après les autres.

La première étape consiste à identifier la lettre.

Si la lettre est présente dans le fichier PDF sous forme de polyligne, alors, la première opération consiste à transformer cette information graphique en une lettre au sens informatique du terme. Pour cela, nous utiliserons un algorithme de reconnaissance de lettres, équivalent aux OCR (*Optical Caracter Recognition*) que nous appliquons aux glyphes du fichier d'entrée (voir paragraphe 4.2 et paragraphe 4.2.2, page 79).

La deuxième étape consiste à identifier les mots.

Dans un fichier PDF, il est très rare de trouver des textes composés de plus d'une lettre. Les lettres d'un mot sont isolées les unes des autres. En général, les travaux de l'état de l'art font des recherches de proximité dans l'espace 2D pour trouver les lettres qui peuvent s'associer les unes avec les autres et former

ainsi un mot.

Notre approche est totalement différente, nous regardons si l'objet graphique précédent, dans le fichier d'entrée, est une lettre et si la réponse est oui, alors on contrôle que la lettre en cours peut s'associer avec les lettres précédentes.

L'étape incrémentale de restructuration des mots consiste à assembler tous les objets graphiques textuels qui représentent des lettres pour les remplacer par l'objet de plus haut niveau qui matérialise le mot (voir paragraphe 4.2.3, page 86).

Remarque : un mot peut être composé d'une seule lettre, mais en règle générale, il contient entre 5 et 10 lettres dans les domaines de la schématique. Ce passage de lettres à mots divise le nombre d'objets graphiques textuels par un coefficient compris entre 5 et 10.

La troisième étape consiste à identifier les phrases.

Les phrases sont des ensembles de mots. Le problème est d'associer correctement les mots les uns avec les autres. Dans les schémas ou les plans, il existe un certain nombre de problèmes supplémentaires et différents par rapport à l'identification des mots et des phrases dans un article de presse ou scientifique. Les mots ont des orientations quelconques dans les schémas et les plans.

Comme pour les mots, les travaux de l'état de l'art font des recherches de proximité dans l'espace 2D pour trouver les mots qui peuvent s'associer les uns avec les autres et former ainsi une phrase. Notre approche est totalement différente, nous regardons si l'objet graphique précédent, dans le fichier d'entrée, est un mot et si la réponse est oui, alors on contrôle que le mot en cours peut s'associer avec les mots précédents.

Les phrases réelles (au sens des traitements de textes) sont très rares en schématique, mais elles peuvent exister. Par contre les phrases, telles que définies ci-dessus, existent dans les folios de nomenclatures des câbles, des matériels, des listes de folios, ou dans les titres des folios.

L'étape incrémentale de restructuration des mots consiste à assembler tous les objets graphiques textuels qui matérialisent des mots pour les remplacer par l'objet de plus haut niveau qui matérialise la phrase (voir paragraphe 4.2.4, page 88).

Remarque : une phrase peut être composée d'un seul mot, mais en règle générale, quand elle existe, elle contient entre 1 et 10 mots ; dans les domaines de la schématique, ce passage de mots à phrase a peu d'effet sur la diminution du nombre d'objets, car les phrases sont peu nombreuses.

La quatrième étape consiste à identifier les paragraphes.

Les paragraphes, au sens des traitements de textes, sont encore plus rares que les phrases. Il est rare que les textes présents sur un schéma ou un plan forment des paragraphes. Les types de documents dans lesquels on trouve des paragraphes sont, par exemple, les notices de montage et les gammes d'usinages. Ces documents expliquent textuellement un enchaînement d'opérations en s'appuyant sur des schémas. Dans ce type de documents, pour identifier les paragraphes, là encore, notre approche est différente de celle de l'état de l'art :

nous regardons si l'objet graphique précédent, dans le fichier d'entrée, est une phrase et si la réponse est oui, alors on contrôle que la phrase en cours peut s'associer avec la phrase précédente.

L'étape incrémentale de restructuration des paragraphes consiste à assembler tous les objets graphiques textuels qui matérialisent des phrases pour les remplacer par l'objet de plus haut niveau qui matérialise le paragraphe (voir paragraphe 4.2.5, page 90).

Remarque : un paragraphe peut être composé d'une seule phrase, et c'est souvent la cas, ce passage de phrases à paragraphe a peu d'effet sur la diminution du nombre d'objets.

En conclusion de la restructuration des objets textuels :

La diminution du nombre des objets dans le fichier d'entrée est moins importante que pour les graphiques. On divise le nombre des objets textuels par un coefficient compris entre 5 et 10 en fonction du document d'entrée. Mais surtout ce sont des étapes incrémentales indispensables pour la suite du processus.

Le cas général de la restructuration des objets textuels, pour de la schématique, donne le résultat suivant : un paragraphe est égal à une phrase, elle même égale à un mot qui, lui, contient de 1 à 10 lettres.

Maintenant que ces travaux préliminaires sont faits et ont diminué la taille du fichier d'entrée, on recherche les symboles et les liaisons (équipotentielles).

3.2.3 La restructuration des symboles et des liaisons

La notion de symboles : dans les schémas d'automatismes et d'électromécaniques, nous avons 2 types de symboles, les symboles électriques (protection, sectionneurs, contacts, voyants, etc...) et les symboles non électriques (essentiellement les cartouches).

Un symbole est composé de 1 à N objets graphiques et/ou textuels. Les symboles électriques sont, par exemple (voir la figure D.2, page 138), la protection, le sectionneur et le moteur. Ces symboles électriques sont reliés à des liaisons (équipotentielles). Quand on a identifié un symbole, on a identifié également les liaisons (équipotentielles) qui sont en entrée et en sortie du symbole.

Un symbole relié à une liaison (équipotentielle) est un symbole électrique. Tous les symboles non reliés à une liaison (équipotentielle) sont des symboles non électriques. Les symboles non électriques sont, par exemple, le cartouche qui est utilisé dans l'exemple de la figure D.2, page 138, ou bien les flèches utilisées pour la cotation des plans des façades des équipements ou des armoires électriques.

La notion de liaison : une équipotentielle est la liaison électrique qui relie deux ou plusieurs symboles électriques. Une équipotentielle est composée de 1 à N segments de droites et de 1 à N objets textuels (en général un objet textuel).

L'identification des symboles : elle consiste à retrouver des objets graphiques et textuels qui se répètent à l'intérieur du document. Si ces objets se

répètent plus d'une centaine de fois dans un dossier, c'est très certainement un symbole. Puis on cherche les symboles qui se répètent plus de 50 fois, puis plus de 20 fois et ainsi de suite.

L'étape incrémentale de restructuration des symboles et des équipotentielles consiste à assembler tous les objets graphiques structurés (dans les étapes précédentes) et non structurés qui représentent ces symboles et ces équipotentielles pour les remplacer, en lieu et place, par l'objet de plus haut niveau qui matérialise ces symboles et ces équipotentielles (voir paragraphe 4.3.1, page 92).

Une fois un symbole identifié, par le fait que les objets se répètent dans le fichier d'entrée, deux cas peuvent se produire :

— soit cette répétition d'objets a déjà été enregistrée dans la base de connaissances : dans ce cas, on la remplace par le symbole que l'on a associé à cet ensemble d'objets

— soit elle n'existe pas dans la base de connaissances : dans ce cas, l'utilisateur détermine si c'est réellement un symbole. Si la réponse est oui, il va l'enregistrer dans la base de connaissances de sorte qu'elle soit réutilisée quand cette association se présentera de nouveau dans un autre document.

3.2.4 La restructuration des fonctions

Une fonction est un ensemble composé par l'association de 0 à N symboles et de 0 à N liaisons (équipotentielles), et c'est un ensemble non vide. L'identification des fonctions se fait de la même façon que l'identification des symboles à la seule différence près suivante :

— pour identifier un symbole, on cherche les objets graphiques et textuels qui se répètent,

— pour identifier les fonctions, on cherche les symboles et les liaisons (équipotentielles) qui se répètent.

L'étape incrémentale de restructuration des fonctions consiste à assembler tous les objets graphiques structurés à l'étape précédente (symboles, équipotentielles) qui représentent ces fonctions pour les remplacer, en lieu et place, par l'objet de plus haut niveau qui matérialise cette fonction (voir paragraphe 4.3.1, page 92).

Une fois une fonction identifiée, on procède exactement de la même façon pour savoir si la fonction est dans la base de connaissances ou s'il faut l'y rajouter.

3.2.5 La restructuration du dossier

La restructuration du dossier consiste à ventiler les fonctions dans les différents folios du dossier électrique puis associer à ces fonctions les titres et sous titres des folios lorsqu'ils existent (ce qui est en général le cas). Après cette opération, le dossier électrique est entièrement restructuré.

La restructuration du dossier est traitée paragraphe 4.3.2, page 95.

3.3 L'utilisateur au coeur de la méthode (A)KDD

Notre approche est différente des travaux actuels, elle est plus graphique et elle met l'utilisateur au centre du processus d'extraction. Nous utilisons bien sûr tous les textes présents dans les fichiers, mais nous utilisons également l'ensemble des objets graphiques qui matérialisent les cartouches, les objets de liaison, les composants, les fonctions, etc. Comme nous venons de le dire, l'utilisateur est présent dans toutes les phases de restructuration. Il est beaucoup moins coûteux et plus efficace de faire intervenir l'utilisateur tout au long du processus (et surtout dès le début, plutôt qu'en fin) pour valider ou corriger les erreurs. Faire prendre les décisions à l'utilisateur, ce qui lui demande quelques fractions de secondes en fonction de l'Interface Homme Machine (IHM) mise en place, et, diminue et/ou annule le taux d'erreur de chaque tâche.

Dans le processus de la restructuration, le nombre de tâches à faire est de l'ordre d'une vingtaine. Si chaque tâche est entièrement automatisée et produit un taux d'erreur de 2 à 3 % (ce qui est très peu,puisque ça implique 98 % à 97 % de taux de reconnaissance exact), on constate qu'en bout de chaîne, pour procéder à la validation, l'utilisateur va devoir prendre des décisions sur des informations qui auront entre 40% et 60% d'erreurs. Les erreurs commises dans les premières tâches entraînent plus d'erreurs dans les tâches suivantes.

Et c'est ce taux d'erreur qui peut entraîner le rejet du système par l'utilisateur. Dans la phase de validation, il va devoir, modifier, reclasser ou annuler plus de 40 % des résultats. Si la donnée déstructurée contient 10 000 informations de base, il devra modifier plus de 4 000 informations. Ces tâches de correction vont lui prendre un temps considérable, inacceptable dans le monde industriel.

Chacune des phases de (A)KDD est découpée en tâches et sous-tâches pour que l'utilisateur puisse intervenir, faire les choix. Le mot « Utilisateur » est un terme générique qui concerne tous les types d'utilisateurs intervenant dans la restructuration, cela va de l'expert au simple utilisateur.

« Un des maître-mots en IHM est le centrage utilisateur. La conception centrée utilisateur est un paradigme définissant 12 principes clés qui mettent l'utilisateur à une place centrale au sein des processus de développement » (Norman et Draper [ND86]). C'est ce paradigme précisé dans « Modélisation en Interaction Homme-Machine et en Système d'Information : à la croisée des chemins » par Dupuy-Chessa [Dup11] que nous utilisons pour définir nos tâches utilisateurs et systèmes. Pour des raisons de clarté, nous ne décrirons que les principales tâches de chacune des phases.

La restructuration est une opération longue et complexe. Elle doit-être exacte à 100 %, c'est pourquoi elle est « Anthropocentrée », guidée par l'utilisateur. Mais il faut que les tâches utilisateurs décrites ci-dessous soient ergonomiques, implicites, simples et efficaces. Surtout celles qui visualisent beaucoup de données.

Pour l'interprétation et l'évaluation, nous reprenons ici les propos de Guy Melançon dans « Visualisation d'information » [MEL15]. « Le succès de l'opéra-

tion de fouille tient à la possibilité de lire les motifs découverts. L'interprétation de ces motifs, l'évaluation de leur pertinence et du potentiel applicatif des résultats de la fouille restent une affaire bien humaine, dont la complexité et la subtilité ne peuvent à ce jour être confiées à un automate. Seul l'humain peut in fine juger de la pertinence d'un résultat, en apprécier le sens et l'impact potentiel, et le cas échéant prendre les bonnes décisions ».

La question est simple : comment visualiser un maximum d'informations sur un poste de travail qui sera utilisé par les services techniques de la « *Smart factory* » dans les 10 prochaines années ? La réponse est plus complexe.

C'est pourquoi nous avons axé notre recherche sur le concept « *Overview first, zoom and filter, then details-on-demande* » énoncé dans l' « *information seeking mantra* » de Shneiderman en 1996 [Shn96]. En respectant la taxonomie TTT (*Type by Task Taxonomy of information visualization*) du même article, et ses sept tâches « *overview, zoom, filter, details-ondemand, relate, history and extract* » que Keim en 2002 [Kei02] reprend dans la définition de « *Visual Exploration Paradigm* ».

Les techniques de « *FishEyesView* » très prometteuses dans certains domaines, comme pour les règles d'association de Chevrin en 2007 dans « Recherche anthropocentrée de règles d'association pour l'aide à la décision » [Che+07] ne nous semblent pas adaptées à notre problème car la possibilité de grossissement est faible (4 à 5 fois la taille originale). Une étude très détaillée de Mikkel Rønne Jakobsen et Kasper Hornbæk montre que, sur les petits écrans (640x267) ou les écrans moyens (1920x800), les méthodess « *Overview + Detail* » sont plus performantes que « *Zooming + Panning* » elles-même plus performantes que « *Focus + Context* » ceci même pour les très grands écrans (5760x2400) [RH11].

C'est le même constat que nous avons fait et qui nous a poussé à développer une méthode qui est un mixte des 2 méthodes « *Overview + Detail* » et « *Zooming + Panning* ». Les modalités et les usages des tâches sont :

— pour certaines figées par le système,
— pour d'autres choisies par le système,
— et enfin pour les dernières choisies par l'utilisateur.

Des techniques de « *eye-tracking* » se développent de plus en plus pour aider l'utilisateur dans ses choix. Dans une étude de 2018, les auteurs de « *Encoding decisions and expertise in the operator's eyes : Using eye-tracking as input for system adaptation* »[Cau+19] concluent sur le fait que « Nos résultats soulignent l'intérêt potentiel de l'*eye tracking* pour améliorer et accélérer l'adaptation du système aux préférences, aux connaissances et aux expertises des utilisateurs ». La mise en oeuvre d'une telle technique dépasse le cadre des objectifs que nous nous sommes fixés pour restructurer les documents.

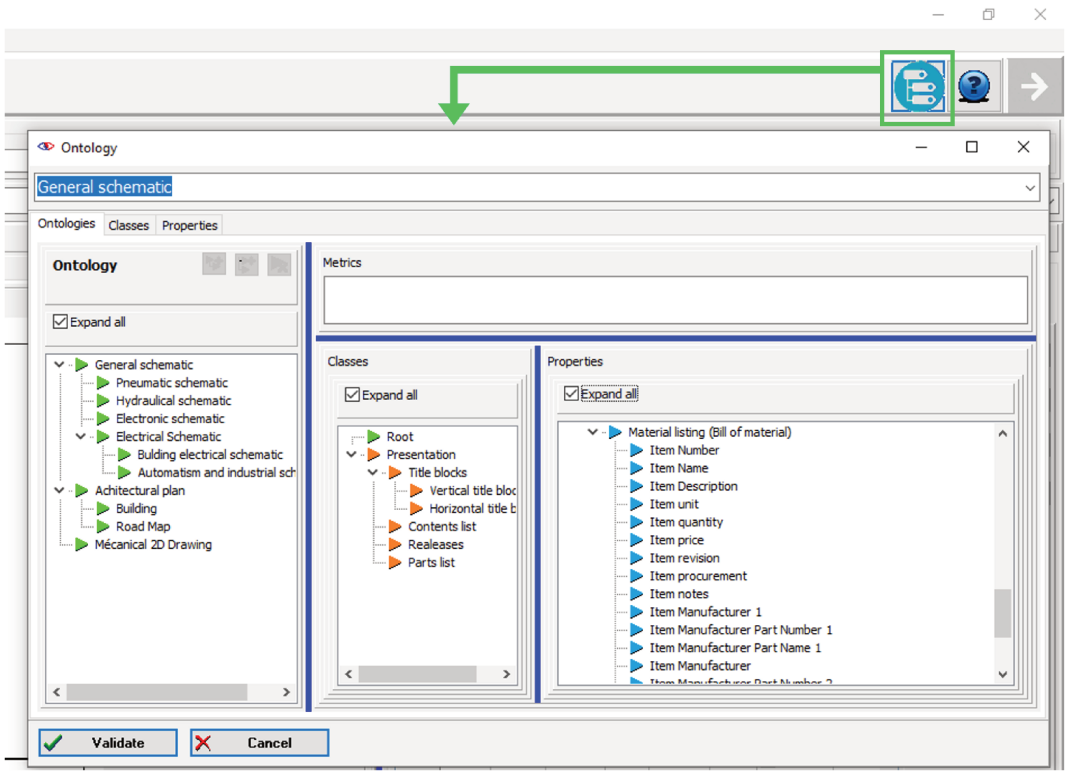


FIGURE 3.7 – Écran de sélection de la base de connaissances et de l'ontologie

3.4 Phase 1 : La sélection et le nettoyage

3.4.1 La sélection

Pour intégrer les données, l'utilisateur définit la base de connaissances et l'ontologie ainsi que la base graphique qu'il va utiliser (par défaut ce sont les dernières utilisées). C'est la première tâche de cette phase. L'utilisateur appuie sur le bouton en haut à gauche de l'écran , pour afficher la fenêtre des ontologies et choisir ainsi l'ontologie à utiliser et la base de connaissances liée à l'ontologie (voir figure 3.7).

La deuxième tâche est encore une tâche utilisateur : il sélectionne les fichiers à traiter. La première liste déroulante permet de sélectionner de 1 à N fichiers à traiter en utilisant la même base de connaissances. Il est possible de les sélectionner directement dans les fichiers du système d'exploitation grâce au bouton situé à droite de la liste déroulante et faire une sélection multiple. La deuxième liste déroulante permet de choisir un des fichiers présélectionnés dans la première liste déroulante. Ensuite l'utilisateur choisit les folios qu'il veut restructurer en les visualisant sur la partie droite de l'écran (voir figure 3.8).

La troisième tâche est une tâche système. Elle consiste à lire le fichier PDF qui a été sélectionné dans la deuxième liste déroulante, et, à extraire de ce fichier toutes les informations graphiques :

— des segments, des polylignes,
— des textes de 1 à n caractères,

— des stylos, des brosses et des fontes.
— etc.

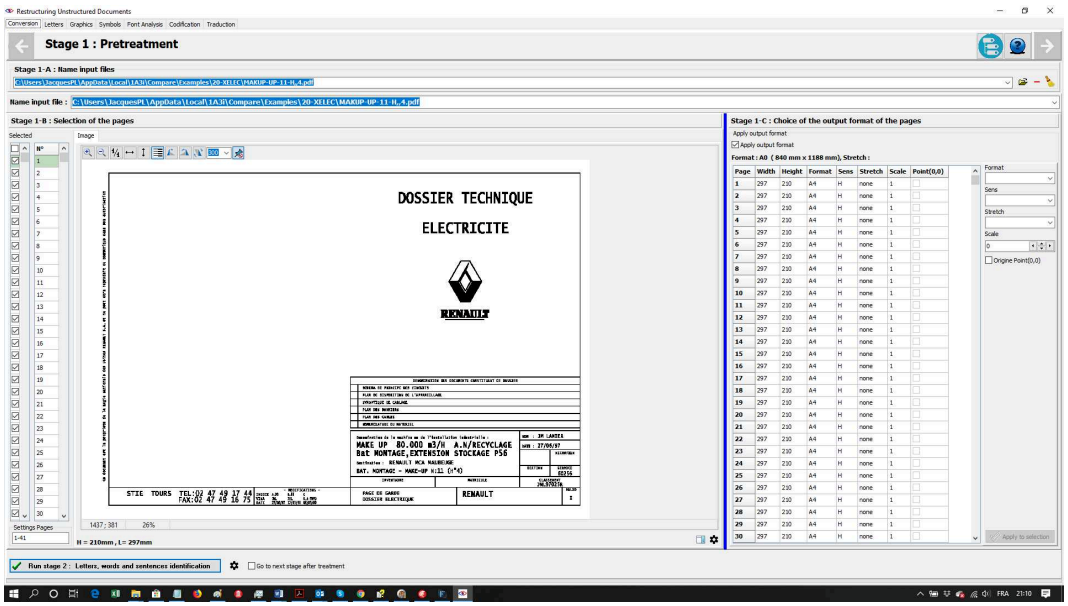


FIGURE 3.8 – Écran de sélection des fichiers et des folios à traiter

Dans le cas de présence d'éléments graphiques plus complexes comme les cercles, les arcs de cercle, les ellipses, les carrés, les rectangles, les triangles , les polylignes, ces éléments graphiques sont conservés tels quels. Ce cas est très rare.

Toutes ces informations graphiques sont converties dans un format pivot. Nous avons choisi le format DXF/DWG (Format AutoCad™), mais également le format EMF. Ces formats sont lus par les principaux logiciels de CAO, DAO et PAO. Le format EMF est utilisé pour introduire les données graphiques dans Word™.

Cette tâche système est plus ou moins complexe en fonction du type de fichier en entrée (assez difficile pour les fichiers PDF). Elle est indispensable à la restructuration des données. C'est une tâche d'ingénierie pure, conçue et développée avant le début de la thèse, voir Annexe D.3, page 142, pour les types de formats autres que PDF.

3.4.2 Le nettoyage

Le nettoyage est une tâche système qui comprend plusieurs opérations. Ce nettoyage est une tâche fondamentale dans la fouille de données. Elle peut représenter une charge très importante. En 2000, Maletic dans « *Data Cleansing : Beyond Integrity Analysis* » [MM00] indique que des entreprises dépensent des millions de dollars pour corriger leurs données « *Some organizations spend millions of dollars per year to detect data errors* ».

Dans nos fichiers, en entrée, il y a beaucoup d'anomalies qu'il faut impéra-

tivement détecter et corriger avant d'appliquer les algorithmes. Ce ne sont pas des données aberrantes comme nous l'avons vu paragraphe 3.1.1, page 34, mais c'est la même donnée qui est en double et qui se superpose. Quand on regarde un fichier PDF on croit que tout est parfait, mais, en réalité il faut corriger toutes les anomalies que nous décrivons ci-dessous afin que les algorithmes se déroulent correctement.

3.4.2.1 Les lettres en double

Dans les fichiers PDF, assez souvent, les lettres sont doublées et il arrive même que la même lettre soit présente jusqu'à 9 fois. Elles ne sont pas toutes à la même position, elles sont décalées de 1 à 2 dixièmes de millimètre les unes par rapport aux autres. Elles sont placées sur une grille carrée de 3 par 3. En effet, pour afficher une police en gras [6], une méthode consiste à afficher plusieurs fois le même caractère en le décalant de quelques dixièmes de millimètres à chaque fois, de cette façon on obtient un caractère gras avec le dessin d'un caractère normal.

Ceci se passe essentiellement sur la page de garde et sur tous les folios où il peut exister des titres (en général en gras).

3.4.2.2 La suppression des tracés en double

Les traceurs (voir note 6) pouvaient avoir de 4 à 10 plumes. Pour des raisons d'optimisation de temps du tracé, quand il y avait un changement de couleur, on ne changeait pas de plume, on faisait comme si le tracé de cette couleur se faisait plume haute. Quand on avait fait une première couleur, on recommençait avec la deuxième couleur en suivant le même principe, et ainsi de suite. Ce qui fait que, dans de nombreux fichiers PDF, on peut retrouver de 4 à 10 fois exactement le même tracé. Ceci multiplie la taille du fichier donc du nombre d'objets. On peut diviser le nombre d'objets en entrée par un coefficient pouvant aller de 4 à 10, ce qui n'est pas négligeable.

3.4.2.3 Les surfaces cachées

C'est le troisième problème à gérer. Il s'agit de fichiers qui ont été repris ou modifiés par une société qui n'est pas la créatrice du dossier. On peut voir des techniques qui permettent de masquer le nom de l'entreprise qui l'a réalisé et de le remplacer par le nom de l'entreprise qui fait la modification.

Pour atteindre ce résultat, la société qui modifie le document place sur le logo et le nom de l'entreprise qui a créé le document, un rectangle avec des bords blancs et un remplissage à blanc. De cette façon le logo et le nom du créateur seront recouverts d'une zone blanche, sur laquelle, la société qui modifie le document, mettra son logo et son nom. Pour être conforme à l'apparence du

6. Les fichiers PDF sont construits à partir des techniques utilisées pour tracer les documents sur table traçante (voir paragraphe A.1.2, page 127).

folio, il faut supprimer les objets qui sont recouverts par une forme dont le remplissage est blanc (un rectangle, un carré, une polyligne).

3.4.2.4 Les liaisons surchargées

Certains logiciels, mais pas tous, permettent que les liaisons (équipotentielles) qui sont composées de segments de droite se recouvrent partiellement ou totalement, car ils traitent cette possibilité de recouvrement dans leur logiciel. Mais les logiciels qui n'ont pas cette fonctionnalité peuvent ne plus fonctionner et détecter des erreurs de connexions.

On peut penser que ce type de recouvrement doit être traité lors du nettoyage du fichier, c'est une erreur que nous avons faite. Il faut traiter ce problème après la détection des liaisons (équipotentielles). Si on le traite au début cela peut avoir des répercussions sur la reconnaissance des symboles, certaines parties du symbole (segment de droite) peuvent être fusionnées avec les liaisons, et disparaître. De plus on casse l'ordre chronologique des objets d'entrée, ce qui est très mauvais pour notre méthode.

3.5 Phase 2 : La codification

La phase de codification est le coeur de nos travaux de recherche et elle repose sur l'hypothèse que nous avons formulée dans le paragraphe 3.1.2, page 35.

Quand on analyse les fichiers PDF et les fichiers vectoriels issus d'un même logiciel, on s'aperçoit que les objets graphiques vectoriels sont dessinés toujours dans le même ordre chronologique. Ceci est vrai pour les objets graphiques de base (les carrés, les rectangles, etc.), mais aussi pour les objets graphiques plus complexes comme les cartouches, les cartes d'automates, etc. Ces informations chronologiques sont contiguës. Pour certains logiciels ce n'est pas toujours le cas, on peut trouver d'abord les objets graphiques de type segments (rangés dans un ordre chronologique bien précis), puis un peu loin dans le fichier, les objets graphiques textuels (également rangés chronologiquement).

La plupart des solutions, pour restructurer les informations contenues dans les fichiers PDF, se servent de la position X, Y des objets et de leur proximité. C'est une solution que nous utilisons, mais uniquement pour lier les textes aux symboles et seulement si c'est nécessaire. Pour la partie graphique vectorielle nous n'utilisons que l'ordre chronologique des données. Et pour ce faire, nous allons coder tous les objets graphiques de base (comme pour un alphabet) et nous pouvons ainsi manipuler des chaînes de « caractères » (de codes) au lieu d'objets graphiques d'un espace à 2 dimensions.

Pour expliquer le principe de la codification et de la transformation de l'espace 2D en un espace 1D, nous allons décrire une codification très simple sur un exemple concret qui est proche de la codification des symboles.

Dans la figure 3.9 nous avons, sur la partie haute, le symbole de la bobine d'un relais qui a pour repère « KM1 ». Ce symbole est composé d'éléments

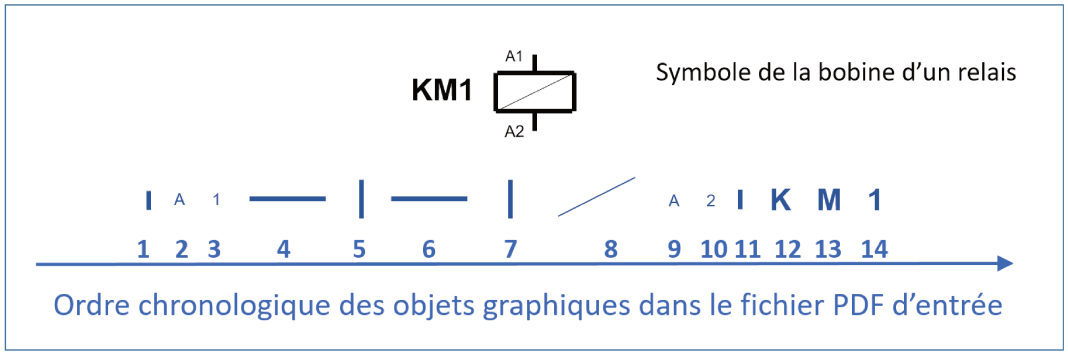


FIGURE 3.9 – Le symbole bobine de relais dans le fichier PDF

graphiques qui sont des segments de droite verticaux, horizontaux et un segment oblique. Ces segments sont longs ou courts, de grandes lettres pour le texte « KM1 » et de petites lettres pour les deux autres textes « A1 » et « A2 ».

En tout, nous avons 14 objets graphiques, 7 segments de droite et 7 lettres. Le premier objet graphique c'est un petit segment vertical, le deuxième la petite lettre « A », le troisième la petite lettre « 1 », puis un grand segment de droite horizontal, et ainsi de suite. Ceci est visualisé sur la figure 3.9.

Si nous décidons d'affecter une lettre à chaque objet graphique, par exemple :
- la lettre « H » ou « h » pour des segments horizontaux (« H » pour les grands, « h » pour les petits).
- la lettre « V » ou « v » pour des segments verticaux (« V » pour les grands, « v » pour les petits).
- la lettre « O » ou « o » pour des segments horizontaux (« O » pour les grands, « o » pour les petits).
- la lettre « T » ou « t » pour une succession de lettres de même taille (« T » pour les grandes, « t » pour les petites).

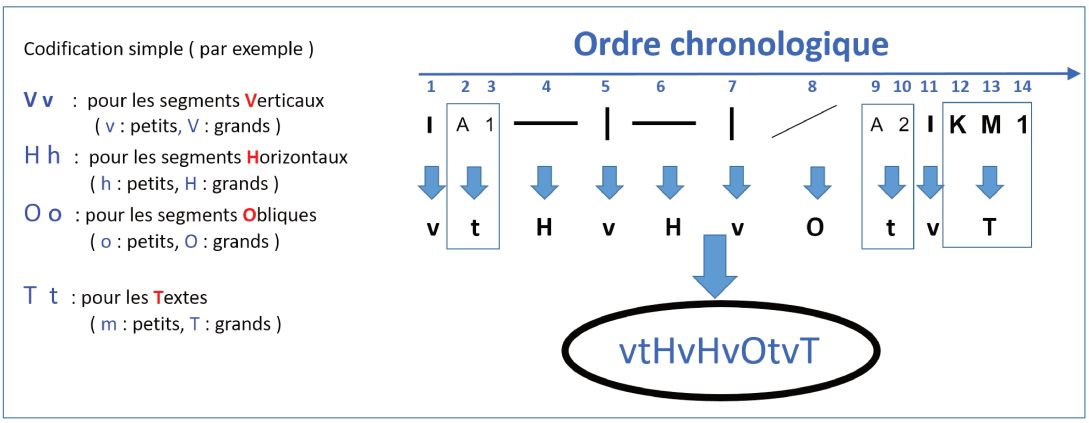


FIGURE 3.10 – La codification du symbole : de la bobine d'un relais

Après l'application de cette codification (très simpliste) aux objets graphiques de la bobine de relais, nous avons le code suivant « vtHvHvOtvT » voir figure 3.10, page 55.

Si la codification que nous venons de décrire et un sous ensemble de la codification utilisée pour coder l'ensemble des objets graphiques du fichier PDF

en entrée de la restructuration, nous pouvons obtenir une chaîne de codes qui est représentée par la figure 3.11.

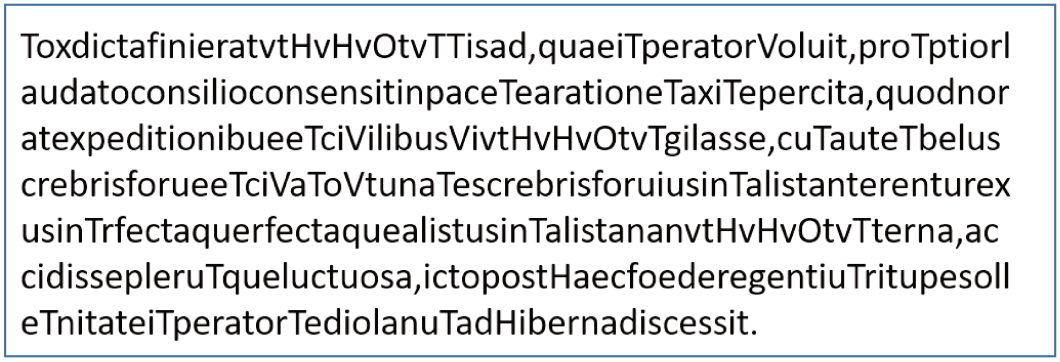


FIGURE 3.11 – Exemple de codification du fichier PDF d'entrée

L'objectif de la fouille de données décrite dans le paragraphe suivant, est d'identifier, dans cette chaîne, le code « vtHvHvOtvT » qui est matérialisé dans la figure 3.12, par le texte en rouge et les flèches entre le symbole de la bobine du relais et les codes présents dans le texte.

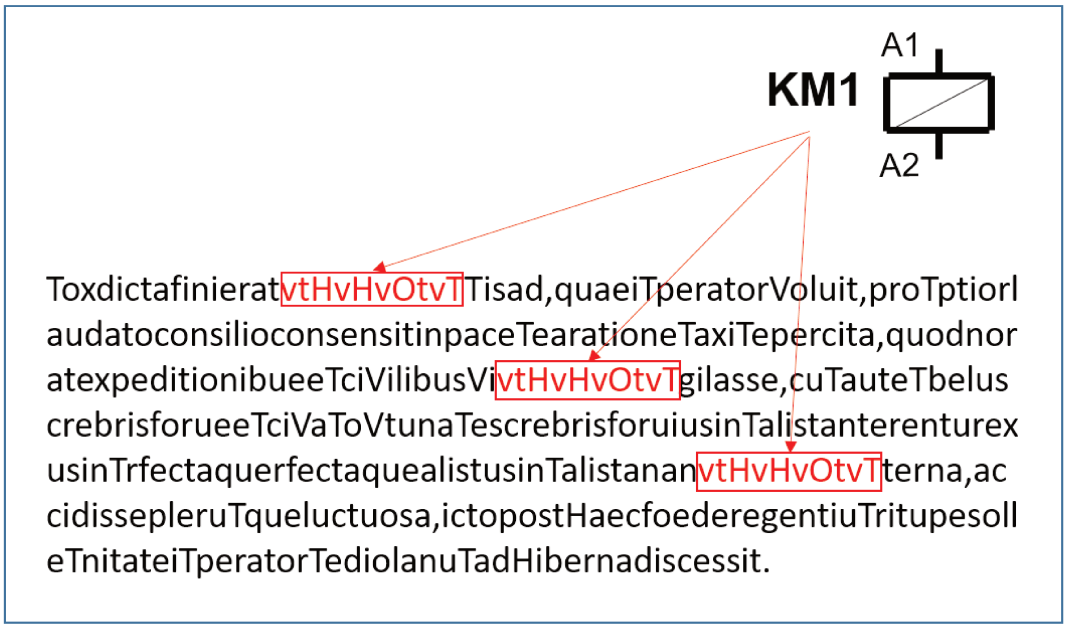


FIGURE 3.12 – Identification du code du relais dans la chaîne d'entrée.

En réalité, nous utilisons des types de codification parfois plus simples et parfois beaucoup plus complexes que dans l'exemple ci-dessus.

La codification est différente en fonction de l'étape incrémentale en cours et de l'objet en cours de restructuration. Toutes ces codifications sont définies dans le chapitre sur le détail des étapes incrémentales :

- Étape 1-1 : codification des traits discontinus, voir paragraphe 4.1.1, page 67
- Étape 1-2 : codification des hachures, voir paragraphe 4.1.2, page 74
- Étape 1-3 : codification des coniques, voir paragraphe 4.1.3, page 76

— Étape 1-4 : codification des formes simples, voir paragraphe 4.1.4, page 78
— Étape 2-1 : codification des lettres, voir paragraphe 4.2.2, page 81
— Étape 2-2 : codification des mots, voir paragraphe 4.2.3, page 86
— Étape 2-3 : codification des phrases, voir paragraphe 4.2.4, page 88
— Étape 2-4 : codification des paragraphes, voir paragraphe 4.2.5, page 90
— Étape 3 : codification des symboles et des liaisons, voir paragraphe 4.3.1, page 92
— Étape 4 : codification des fonctions, voir paragraphe 4.3.2, page 95
— Étape 5 : codification du dossier, voir paragraphe 4.3.3, page 96

Pour des raisons de clarté et de compréhension de chaque étape incrémentale, nous avons regroupé les 3 phases (la codification, la fouille et l'interprétation-évaluation) dans l'étude détaillée de chaque étape incrémentale de la méthode (A)KDD.

3.6 Phase 3 : La fouille

La fouille de données pour la restructuration se décompose, elle aussi, en plusieurs étapes. Ces étapes sont les étapes incrémentales de la méthode (A)KDD :

Étape 1 : elle est relative à la restructuration des objets graphiques vectoriels. Restructurer les traits discontinus : c'est définir les types de lignes et les identifier à partir de leur composition (longueur des traits et des espaces). Restructurer les remplissages et les hachurages : c'est remplacer une multitude de segments de droite par un objet rempli ou hachuré. Restructurer les coniques : c'est remplacer des segments de droites par la forme géométrique qui matérialise la conique, cercle, ellipse, arc de cercle, arc d'ellipse. Restructurer les formes simples : c'est remplacer des segments de droites par une forme géométrique qui matérialise ces formes simples, carré, rectangle, parallélogramme, triangle.

Étape 2 : elle est relative à la restructuration des objets textuels. Restructurer les lettres : (certains logiciels dessinent directement les lettres des mots) c'est définir la police, la taille, le style, l'orientation et les lettres qui associent un ou plusieurs graphismes, par exemple « e », « é », « ë» à un code représentant le caractère dans un alphabet. Restructurer les mots : c'est regrouper les lettres identifiées pour former des mots. Restructurer les phrases : c'est regrouper les mots identifiés pour former des phrases. Restructurer les paragraphes : c'est regrouper les phrases identifiées pour former des paragraphes.

Étape 3 : restructurer les symboles de base : c'est définir ces symboles dans la base de connaissances s'ils n'existent pas, et associer des objets graphiques de base et des textes.

Étape 4 : restructurer les fonctions : c'est définir ces fonctions dans la base de connaissances si elles n'existent pas, et regrouper les symboles simples et les liaisons qui réalisent une fonction. Cette étape est également récursive, une fonction est composée de sous fonctions, elle-même composée de sous fonctions, etc. . . .

Étape 5 : restructurer le dossier : c'est définir les grands domaines et sous

domaines dans lesquels nous regroupons les fonctions.

La fouille des données, comme pour la codification, est expliquée dans le détail de chaque étape incrémentale de la méthode (A)KDD (voir du paragraphe 4.1.1 au paragraphe 4.3.3, page 67).

Les principales techniques que nous utilisons pour faire la fouille de données sont les recherches d'un motif qui se répète localement (comme pour les traits, les hachures, les cercles, les mots, les phrases, les paragraphes) ou qui se répète sur l'ensemble du fichier d'entrée (comme les symboles, les fonctions).

Pour cela, nous avons développé une méthode de recherche des répétitions qui se rapproche de la solution de la création du tableau des suffixes et de la recherche des répétitions à partir de ce tableau des suffixes (voir les méthodes développées par Puglisi [PSY10] et [PST07]). Elle est utilisée pour chercher les répétitions des symboles et des équipotentielles qui sont au coeur du problème de la restructuration des documents déstructurés de type schéma électrique.

3.6.1 Méthode : SAP (*Suffixe Array for Pattern*)

Cette partie a été présentée lors de la conférence EGC de 2019 à Metz (Péré-Laperne [Pér19b]).

Nous partons de la chaîne de départ (issue de la phase 2 de codification), de longueur n, notée S : S est une séquence de n codes de l'ensemble Σ, dans notre cas $|\Sigma| \leq 256$. Pour identifier un symbole, dans un schéma, à partir de la chaîne de codes 1D, nous devons trouver les sous-chaînes de codes qui se répètent au moins T_{inf} fois et qui sont composées d'au moins L_{inf} codes. Prenons deux exemples. Le premier, pour identifier les cartouches dans un dossier électrique de 100 folios, le cartouche est présent sur chaque folio, donc si on cherche les sous-chaînes qui se répètent plus de 90 fois (T_{inf}=90) et qui sont composées d'au moins 50 codes (L_{inf}=50 un cartouche est composé d'un très grand nombre de codes voir Fig.1.2), on est sûr d'avoir identifié les cartouches. Le deuxième pour identifier les protections. Le symbole protection, dans un tel dossier de 100 folios, est présent au moins une trentaine de fois et le nombre de codes qui le composent est d'une quarantaine (T_{inf}=30 et L_{inf}=40). Donc, si on identifie les sous-chaînes qui se répètent au moins 30 fois, et qui ont une longueur supérieure à 40, on est sûr d'identifier les protections.

Dans les travaux de l'état de l'art, le tableau SA est créé pour obtenir la totalité des suffixes triés, puis le tableau LCP permet de déterminer le nombre de codes identiques entre 2 suffixes triés. Ensuite les autres tableaux servent à identifier les répétitions (voir l'état de l'art sur les suffixes array). Dans ces méthodes, ce qui prend le plus de temps, c'est la création du tableau SA car tous les suffixes sont triés.

Nous appelons cette méthode SAP pour *Suffix Array for Pattern*

3.6.2 L'algorithme SAP

L'algorithme que nous proposons ne trie pas tous les suffixes, ne crée pas le tableau (SA), et, il extrait les répétitions en une seule passe. Notre algorithme repose sur l'algorithme de tri par paquets (*bucket sort*, voir algorithme 1). L'article de KÄRKKÄINEN et al. [KR09] décrit ce type de tri qui est linéaire en temps. Pour des raisons de performance SEDGEWICK et al. [SW15] montrent qu'il faut passer au tri par insertion, quand le nombre de suffixes du paquet est proche de $\sqrt{|\Sigma|}$.

Algorithm 1 TriPaquet(R,p)

If $|R| < t$ **then** $insertionSort(Rpd)$ {tri par insertion si nb suffixe < t}
For $s \in R$ **do** $B[S[p]] := B[S[p]U\{s\}$ {range les suffixes dans les paquets}
For $c \in \Sigma$ **do if** $|B[c]| > 0$ **then** $TriPaquet(B[c], p+1)$ {appel récursif}

Ils affirment que c'est le problème majeur de presque tous les algorithmes de tri (*Small subarrays are of critical importance in the performance of MSD string sort. We have seen this situation for other recursive sorts quicksort and mergesort*). Donc, si on supprime le tri des paquets les plus petits, on diminue de façon considérable le temps de tri, c'est l'objectif de notre algorithme.

Avant de le décrire, il faut définir la relation qui existe entre les répétitions et les paquets du tri. Une répétition est un ensemble de sous-chaînes répétées. Une répétition dans S est définie par $R_{S,u} = (L_r; i_1, i_2..., i_{T_r})$, où $T_r \geq 2$ et $0 \leq i_1 < i_2... < i_{T_r} \leq n-1$, la longueur de L_r est égale à la longueur des sous-chaînes, la taille T_r est égale au nombre de sous-chaînes et les i_i sont les adresses des sous-chaînes. Pour l'algorithme de tri, un paquet est égal à la répétition $R_{S,u} = (p+1; i_1, i_2..., i_{T_r})$ où p est la profondeur de la récursivité et les i_i sont les adresses suffixes du paquet à trier et T_r le nombre de suffixes du paquet. On démontre (assez facilement) que toutes les répétitions correspondent à des paquets et que tous les paquets sont les répétitions de la chaîne S.

Algorithm 2 SAP(R,p)

$bProfond := True$ {Indicateur de profondeur est mis à vrai}
For $s \in R$ **do** $B[S[p]] := B[S[p]U\{s\}$ {Range les suffixes dans les paquets}
For $c \in \Sigma$ **do** {Pour chaque code : si le paquet n'a pas assez de suffixes}
if $(|B[c]| \geq T_{inf}) and (p \leq L_{inf})$ **then** {Et si profondeur < limite}
  $SAP(B[c], p+1)$ {alors appel récursif}
if $ConditionsPourEnregistrer$ **then** {Si les conditions sont vraies}
  $EnregistreRepetition$ {Alors on enregistre la répétition}
$bProfond := False$ {Indicateur du paquet le plus profond est mis à faux}

Dans notre algorithme (voir algorithme 2, l'appel de la récursivité n'est exécuté que si le nombre de suffixes du paquet est $\geq T_{inf}$ et si la profondeur de la récusivité est $\leq L_{inf}$. L'indicateur $bProfond$ est mis à vrai en début de procédure, et, remis à faux en fin de procédure.

En fin procédure, l'enregistrement des répétitions ne se fait que si $ConditionsPourEnregistrer$ est vrai c'est à dire si $bProfond$ est vrai et si le nombre de suffixes du paquet est $\geq T_{inf}$ et la profondeur p est $\geq L_{inf}$ et à condition que les suffixes ne se chevauchent pas (les suffixes qui se chevauchent sont supprimés du nombre des suffixes du paquet).

Dans notre algorithme SAP, le temps pour identifier les répétitions qui ont un nombre de suffixes supérieurs à 16 (T_{inf}) et une longueur de sous-chaîne supérieure à 8(L_{inf}) est de 0,115 microsecondes (voir figure 5.4, page 101). Ce temps est à comparer aux 1,112 microsecondes du tri complet de tous les suffixes (voir figure 5.3, page 100). Le temps de traitement est divisé par 10. Maintenant, Comparé à ceux de PUGLISI et al. [PSY10] il est 19 fois plus rapide (voir figure 5.2, page 99). L'espace mémoire occupé par notre algorithme est légèrement supérieur à ceux utilisant les tableaux des suffixes, il faut rajouter l'espace occupé par les compteurs : dans notre cas, 1024 octets ainsi que 4 fois la taille du plus grand paquet. On peut diminuer l'espace total occupé à 8 fois la taille du plus grand paquet plus la taille des compteurs, cela complexifie l'algorithme et diminue ses performances d'environ 10% (voir algorithme 3, page 61).

La méthode (A)KDD est centrée utilisateur, c'est lui qui fixe les seuils T_{inf} et L_{inf}. La stratégie consiste à identifier en premier les symboles présents le plus grand nombre de fois et/ou les plus grands, puis, ceux qui apparaissent le moins souvent et/ou les plus petits. A chaque itération, les symboles identifiés sont supprimés du fichier d'entrée. De cette façon l'itération suivante se fait sur un fichier de taille inférieure, elle est plus rapide.

3.6.3 L'algorithme SAP optimisé

Maintenant, nous présentons, l'algorithme optimisé (et un peu plus complexe) pour diminuer l'espace mémoire utilisé par l'algorithme 2, page 59. Cet algorithme optimisé permet de diminuer, de façon très importante, la taille mémoire utilisée pendant l'exécution de l'algorithme, moyennant un temps d'éxécution supérieur d'environ 10% (voir figure 5.5, page 102).

L'algorithme basique décrit précédemment (algorithme 2, page 59) subit les modifications suivantes.

Pour éviter une mauvaise utilisation de la mémoire cache on crée, comme le suggère KÄRKKÄINEN et al. [KR09], un tableau des codes à trier (CodeTri). Lors de la première itération, il est égal à S (donc aucune place supplémentaire n'est nécessaire), et sa taille est calculée pour contenir tous les suffixes du plus grand paquet. Le tableau des suffixes SA contient l'adresse, dans la chaîne S, du début des suffixes du paquet à trier. Le tableau des codes à trier contient le P ième code des suffixes. Comme les auteurs cités, nous constatons une amélioration (voir nos résultats).

L'appel de la récursivité n'est exécuté que si le nombre de suffixes du paquet est $\geq freq_{min}$. Rappel : lors de la construction du tableau des suffixes (SA) ou de l'arbre des suffixes, il faut trier tous les suffixes, ce n'est pas notre cas.

Algorithm 3 SAPoptimise(S :Array of code ; $Freq_{min}$, P_{min},C_{min} : Integer)

procedure Repetitions (lo, hi, p, C_t : integer ; SA : array of code) ;
$bDroite := True$ {Indicateur du panier le plus à droite mis à vrai}
for i :=0 to cSizeCpt do $Cpt[i] := 0$ {Initialisation des compteurs des codes}
if $(hi - lo + 1 >= Freq_{min}) and (p <= P_{min})$ **then**
begin {On prépare les appels récursifs}
if $P = 0$ **then** {Si c'est la première itération}
begin
for i :=lo to hi do $Cpt[s[i] + 1] := Cpt[s[i] + 1] + 1$ {On compte}
$MaxS := 0$ {Calcule la fréquence maximun $MaxS$}
for i :=0 to cSizeCpt do $MaxS := MAX(MaxS, Cpt[i])$ {Boucle}
$Setlength$ de $lSA, lSATmp, CodeTri$ à $MaxS$ {Alloc. lSA, lSATmp}
Nb :=MaxS ; lmax :=-1 {Calcul des lots $\Sigma Cpt[i] \leq MaxS$}
for i :=0 to cSizeCpt do {On calcule LotD, LotF}
-> If (Cpt[i]+Nb <= MaxS) then Nb :=Nb+Cpt[i] ; LotF[lmax] :=i-1
-> else Inc(lmax) ; Nb :=Cpt[i] ; LotF[lmax] :=i-1 ;LotD[lmax] :=i-1
for i :=0 to cSizeCpt $Cpt[i] := Cpt[I] + Cpt[i - 1]$ {Cumul }
for l := 0 to lmax do {Boucle pour préparer lSA, puis appel récursif}
-> FillChar(lSA[0], (MaxS) * sizeof(Integer), 0) ; {Initialisation lSA}
-> LInf := Cpt[LotD[l]] ; {Conservation du debut des codes }
-> for j := Lo to Hi do {Rangement des suffixes dans lSA }
—-> if (s[j]>=LotD[l]) and (s[j]<=LotF[l]) then {dans les limites}
———-> lSA[Cpt[s[j]]-LimiteInf] :=J ; Cpt[s[j]] :=Cpt[s[j]]+1 ; {range}
-> for i :=LotD[l] to LotF[l] do {appel récursif }
—->$Repetition(Cpt[I - 1] - LInf, Cpt[P, I] - 1 - LInf, p + 1, lSA)$;
end
else {Sinon c'est la cas général : ce n'est pas la première itération}
begin
for i :=lo to hi do CodeTri[i] :=S[SA[i]+p] {Préparation de CodeTri }
for i :=lo to hi do Cpt[CodeTri[i]+1] :=Cpt[CodeTri[i]+1]+1
for i :=0 to cSizeCpt $Cpt[i] := Cpt[I] + Cpt[i - 1]$ {Cumul }
for i := lo to hi do {ranger la position dans lSATmp}
—> $lSATmp[lo + Cpt[CodeTri[i]]] := SA[i]$; {dans CodeTri}
—> $Cpt[CodeTri[i]] := Cpt[CodeTri[i]] + 1$ {Le code=le Cpt}
$Move$(SATmp[lo],SA[lo],(hi-lo+1)*Sizeof(Integer)) {Copie }
for i :=0 to cSizeCpt {Pour chaque Cpt : appel récursif }
—> $SAPoptimise(Cpt[i], Cpt[i + 1] - 1, p + 1, C_t + C_i[i], lSA)$
end
end
else
begin
if $ConditionsPourEnregistrer$ **then** {Si ok}
$EnregistreRepetition(SA)$ {Alors on enregistre la répétition}
end
$bDroite := False$ {Indicateur du panier le plus à droite mis à faux}

Nous inversons la boucle de parcours du tableau des codes à trier et du tableau des compteurs. Cette inversion a 2 avantages très importants :

— Nous ne sommes pas obligés de créer un tableau des suffixes (taille $4n$), comme le fait l'algorithme de tri. Nous créons le tableau de suffixes limités lSA dont la taille correspond au nombre de suffixes du panier le plus grand, ce calcul se fait à la première itération. Dans nos exemples d'extraction des symboles, la taille du plus grand paquet est toujours $\leq n/32$. Notre tableau des suffixes limités(lSA) nous permet de passer de $4n$ à une taille $\leq n/8$, puisque l'adresse du suffixe occupe 4 octets.

— Nous enregistrons séquentiellement, et de façon optimale, les suffixes dans ce tableau réduit des suffixes. Pour cela, nous créons des lots de paniers, lors de la première itération, ce qui utilise de façon optimale les mémoires caches et qui compense le temps pris pour parcourir plusieurs fois le tableau des codes à trier (CodeTri).

On rajoute le calcul de l'indice de confiance total C_t qui est incrémenté de l'indice de confiance de la lettre du paquet à trier et passé comme paramètre à chaque itération de la procédure. On rajoute la variable globale booléen $bDroite$ qui indique que l'on empile un paquet. Cet indicateur est mis à 'vrai' lors d'un appel de la récursivité et à 'faux' à la fin de la récursivité.

En fin de procédure récursive on n'enregistre une répétition que si :

— la variable $bDroite$ est vraie (répétition la plus à droite) .

— la période de la répétition est $\geq p_{min}$.

— l'indice de confiance est $\geq C_{min}$.

— les occurrences ne se chevauchent pas.

3.7 Phase 4 : L'interprétation et l'évaluation

3.7.1 Les ontologies

La restructuration des documents est très hiérarchisée. Elle est décrite par une ontologie qui est créée ou adaptée par l'utilisateur à partir des ontologies générales ou des parties d'ontologies qui servent de modèle. L'ontologie se créé tout le long du processus de (A)KDD.

Prenons par exemple le cas d'une usine et des documents techniques dont elle dispose. Nous allons, par exemple, trouver :

— les documents des méthodes,

— les documents de production,

— les documents de maintenance,

  — les plans des bâtiments,

  — les plans des processus industriels,

  — les plans de circulation des fluides,

    — les plans hydrauliques,

    — les plans pneumatiques,

    — les plans électriques,

      — les plans électriques des bâtiments,

- les plans électromécaniques et d'automatismes des machines,
- ceux réalisés par le logiciel AutoCad™,
- ceux réalisés par le logiciel Elec'View™,
- ceux réalisés par le logiciel See Expert™,
- etc...,
- etc...,
- etc...,
- etc...,

En mettant la base de connaissances au plus proche de la sous nature de documents on a de meilleurs taux de restructuration. Les informations que l'on propose à l'utilisateur sont plus fiables. Si les symboles de base à restructurer sont les mêmes pour AutoCad™, Elec'View™, See Expert™, il n'en est pas de même pour les cartouches, les cartes d'entrées ou de sortie d'automates, les variateurs, les cartes spécifiques, etc ... Ces données graphiques diffèrent d'un logiciel à l'autre. De même, les objets à restructurer dans un plan d'architecture sont bien différents de ceux d'un schéma d'automatisme ou d'un schéma pneumatique. Les ontologies sont donc différentes suivant les domaines. Par exemple, les liaisons dans le bâtiment seront les murs et les cloisons alors que dans la schématique d'automatisme ce sont des équipotentielles.

Pour une meilleure identification des mots, des dictionnaires de mots et des structures de mots sont définis dans l'ontologie du domaine. Par exemple, le mot « MOTEUR » peut être pré enregistré dans l'ontologie comme instance de mot existant pour le domaine. Si le mot « MOTEUR » n'est pas une instance, et s'il est présent dans le dossier et validé par l'utilisateur, il devient une instance dans le dictionnaire, et sa fréquence d'apparition augmente. Ceci améliore la reconnaissance des mots car le mot « MOTEUR » aura un indice de confiance plus ou moins fort en fonction du nombre de fois où il apparaît.

C'est la même chose pour des structures des mots. Par exemple, en schématique électrique, le nom des contacteurs de puissance commence par les 2 lettres « KM » et est suivi de 1 à 3 chiffres, par exemple, « KM21 ». On voit bien qu'en utilisant ces structures on peut lever le doute sur le dernier chiffre entre le chiffre «1» et les lettres « I » ou « l ». Ces structures sont créées et enregistrées dans l'ontologie comme instance des structures de mots. Ainsi, le même indice de confiance est utilisé pour les mots qui correspondent à une structure. De plus, on peut associer une structure à un paramètre d'un symbole comme, par exemple, le paramètre qui correspond à sa référence.

Dans l'ontologie, sont définies des familles et des sous familles pour créer les classes des objets et les objets manipulés par l'ontologie. Pour chaque classe ou objet, on précise également les attributs qui leurs sont associés. En général, ces attributs correspondent à des zones de textes proches du symbole. De même, on définit, pour les classes et les objets qui matérialisent des symboles électriques, les points de connexions du symbole sur lesquels les liaisons (équipotentielles) vont venir se raccorder.

3.7.2 Les bases de connaissances

Les bases de connaissances sont les ontologies dans lesquelles on a enregistré les instances différentes des objets graphiques que l'on a trouvés dans les fichiers d'entrée.

3.7.3 Les bases graphiques

Les bases graphiques sont la transcription des instances enregistrées dans la base de connaissances, dans des fichiers compréhensibles par les logiciels de DAO, CAO et PAO. Nous avons choisi de créer ces bases graphiques dans le format DXF/DWG et le format EMF/WMF.

3.7.4 Les interactions

Pour chaque étape de la phase 3 (fouille de données), c'est l'utilisateur qui fait l'évaluation et la validation des informations qui lui sont présentées. Cette phase se décompose en 3 grandes tâches pour chaque étape :

— une tâche système/utilisateur qui présente l'ensemble des informations issues de la tâche système de l'étape considérée et permet à l'utilisateur d'interagir,
— une tâche utilisateur qui sélectionne et introduit l'information dans la base de connaissances,
— une tâche système qui recalcule toutes les informations en tenant compte de l' « objet » identifié, et reboucle sur la première tâche.

L'une des problématiques dans l'Interface Homme Machine est le volume des données à visualiser. Les principales questions sont :

— comment les visualiser ?
— comment naviguer dans ces données ?
— comment valider ou pas les informations ?

L'énoncé du problème est simple mais la solution est plus complexe. Dans les deux articles que nous avons publiés [PC17] et [Pér18b], nous avons dit que la solution est dans un mixte des 2 méthodes « Overview+Detail » et « Zooming+Panning ».

Pour chaque étape incrémentale, comme pour la fouille et la codification, nous décrivons dans les chapitres sur les étapes incrémentales détaillées (du chapitre 4.1.1 au chapitre 4.3.3, page 67) les tâches pour l'interprétation et l'évaluation des résultats.

Section 4

Les étapes incrémentales détaillées

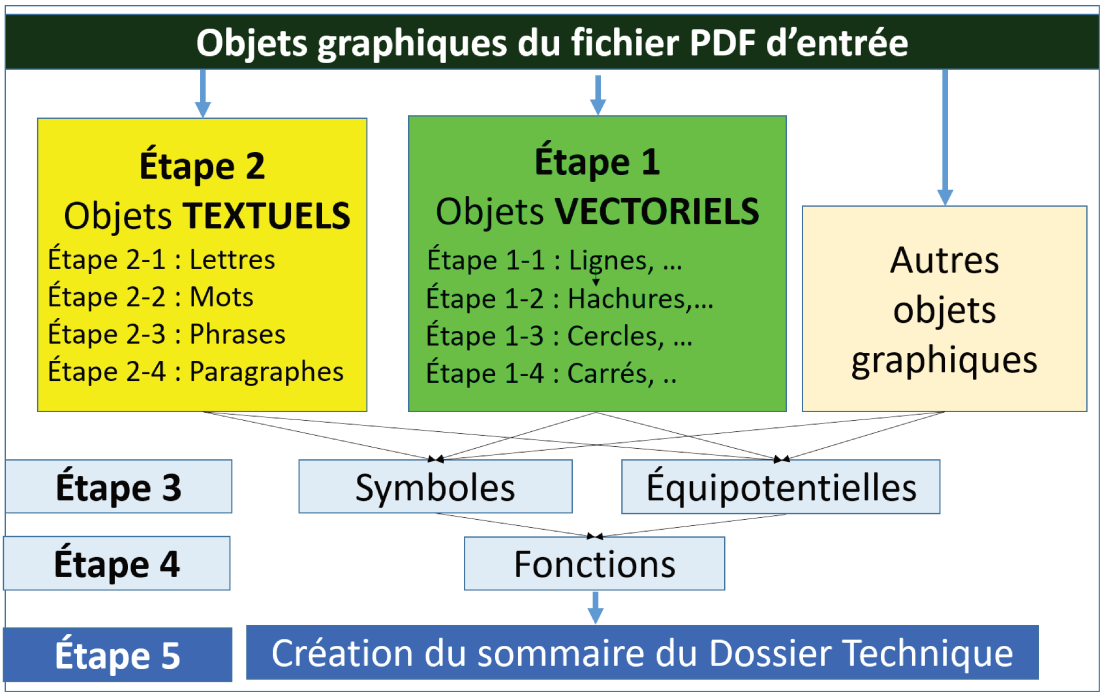


FIGURE 4.1 – Les étapes détaillées de la restructuration

Dans ce chapitre nous allons détailler toutes les étapes nécessaires à la restructuration d'un schéma d'automatisme. Pour rappel, nous avons les étapes de restructuration listées ci-dessous (voir figure 4.1).

L'étape 1 restructure les objets graphiques vectoriels, c'est à dire la restructuration des lignes en traits discontinus, la restructuration des hachures et des remplissages, la restructuration des cercles, ellipses, arcs de cercles, arcs d'ellipses, et, la restructuration des formes simples comme les triangles, carrés, rectangles.

L'étape 2 restructure tous les éléments graphiques textuels pour obtenir en premier les lettres, puis les mots, ensuite les phrases et, pour terminer, les paragraphes.

L'étape 3 restructure les symboles et les équipotentielles : pour cela nous

prenons tous les objets que nous avons restructurés et tous les autres pour trouver les symboles et les équipotentielles.

L'étape 4 assemble les symboles et les équipotentielles pour former les fonctions.

L'étape 5 crée le sommaire du dossier et associe les fonctions avec les lignes du sommaire.

4.1 Étape 1 : Restructuration des objets vectoriels

Nous commençons par restructurer les lignes en traits discontinus car ces lignes peuvent être utilisées pour former les hachures ou les remplissages (voir figure 4.2). Mais elles sont également utilisées pour former les formes géométriques comme les cercles, les ellipses, les carrés, etc. Ensuite nous restructurons les hachures et les remplissages car eux aussi peuvent être réutilisés lors de la création des formes géométriques fermées comme les cercles, les ellipses, les carrés, etc. Et nous terminons par les formes coniques (cercles, ellipses, arcs de cercles, arcs d'ellipses) puis par les formes géométriques simples (triangles, carrés, rectangles, losanges, parallélogrammes). Et si des hachures ou des remplissages correspondent à des formes restructurées, nous associons ces hachures et ces remplissages à ces formes.

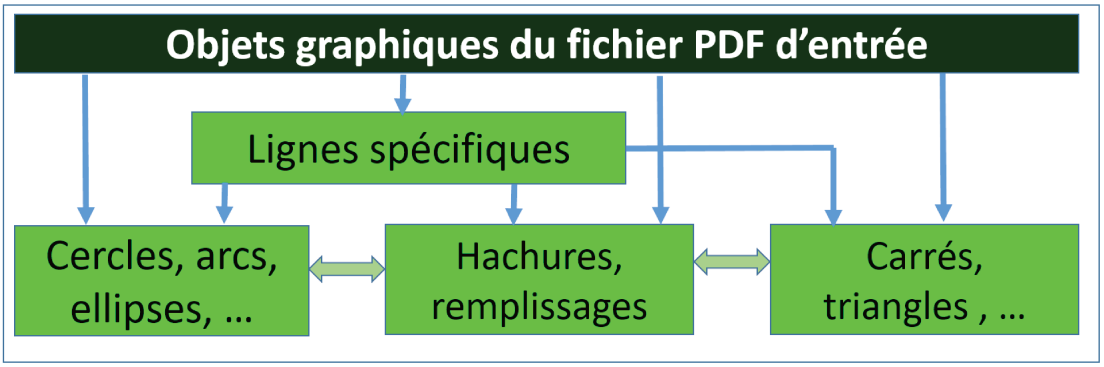


FIGURE 4.2 – Curseurs de définition des lignes discontinues

4.1.1 Étape 1-1 : Restructuration des lignes en traits discontinus

La restructuration des lignes discontinues est une restructuration locale, c'est à dire qu'elle est limitée à un certain nombre d'objets graphiques qui sont des segments contiguës. Bien sûr, les traits discontinus sont présents dans tout le fichier, mais la détection d'un trait discontinu est locale.

4.1.1.1 La codification

Les paramètres

La première sous-tâche est une tâche utilisateur, l'utilisateur fixe trois paramètres :

— le paramètre de la dimension du segment de droite le plus long qui peut entrer dans la composition d'un trait discontinu : la valeur par défaut, prise par le système, est de 15 millimètres, l'utilisateur peut changer cette valeur (elle sera proposée lors du prochain traitement),

— le paramètre de l'espace le plus grand entre deux segments de droite : la valeur par défaut, prise par le système, est de 5 millimètres, l'uti-

lisateur peut changer cette valeur (elle sera proposée lors du prochain traitement),

— le paramètre du nombre de répétitions minimal d'un motif pour considérer que c'est un trait discontinu : la valeur par défaut, prise par le système, est de 2. Le principe de modification de ce paramètre est le même que les deux précédents.

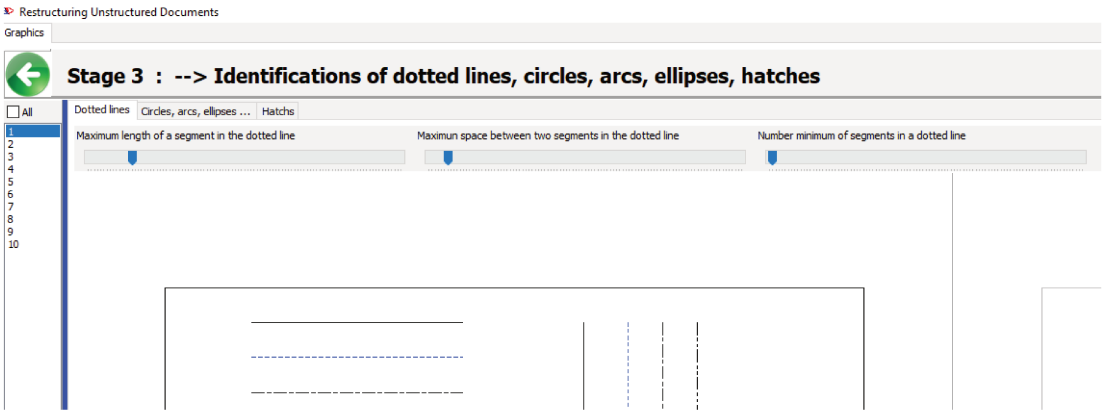


FIGURE 4.3 – Curseurs de définition des lignes discontinues

L'utilisateur fixe ces paramètres avec trois curseurs (voir figure 4.3). Sur l'écran, en superposition de l'image, s'affichent deux carrés concentriques, dont les tailles correspondent à la longueur du segment de droite le plus long (premier curseur : carré rouge), et à l'espace le plus grand (deuxième curseur : carré bleu). De cette façon, ces longueurs sont visualisées voir (figure 4.4). Cette visualisation permet à l'utilisateur de bien se rendre compte de ce qu'elles représentent par rapport aux traits discontinus du dessin.

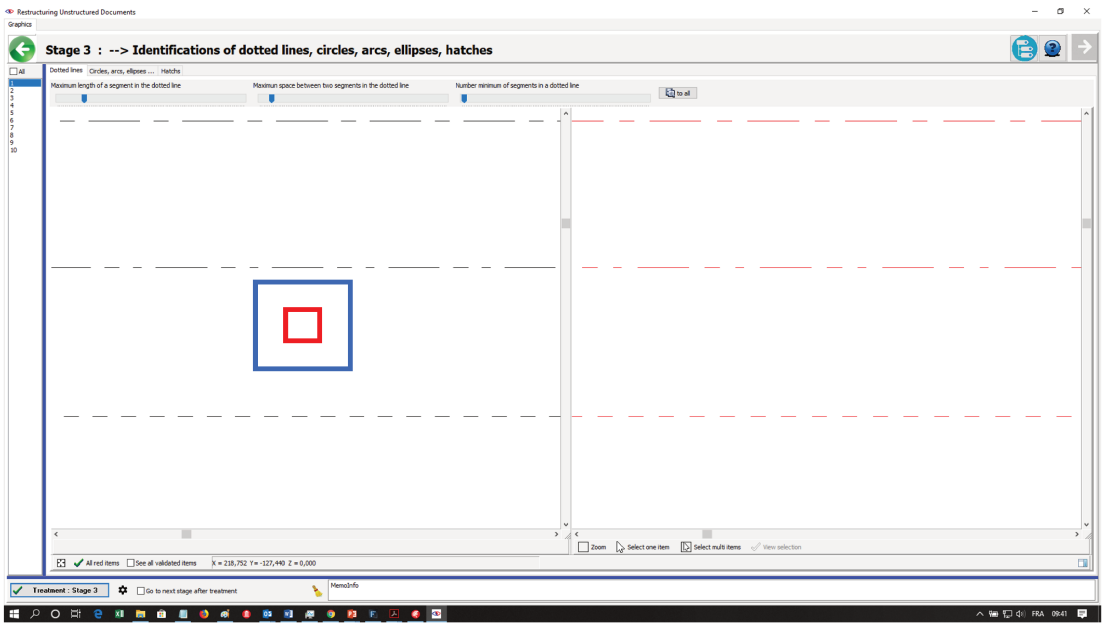


FIGURE 4.4 – Visualisation des valeurs des curseurs par des carrés

Le bouton « *To all* », situé à la droite des trois curseurs, permet d'affecter les valeurs de ces trois curseurs à l'ensemble des folios du dossier.

Le tampon local

La deuxième sous-tâche est une tâche système qui codifie les objets graphiques du fichier d'entrée. Tout objet qui n'est pas un segment de droite reçoit le code 0. Tout segment de droite qui est supérieur à la longueur fixée par l'utilisateur reçoit le code 0. Tout segment de droite qui n'est pas suivi d'un segment de droite de même angle reçoit le code 0.

Pour chaque objet graphique du fichier d'entrée, deux cas peuvent se produire :

S'il est différent de zéro : on calcule le code de ce segment de droite en tronquant le calcul de la longueur du segment de droite multiplié par 10, puis divisé par la valeur du curseur indiquant la longueur maximum d'un segment, et en ajoutant 1 au résultat. Ce code est enregistré dans un tampon. Trois situations peuvent se produire :

- soit le tampon est vide, dans ce cas on l'ajoute au tampon de sortie,
- soit le tampon contient des objets, deux cas peuvent se présenter :
  - le point de fin du dernier segment du tampon est égal au point de début du segment de droite en cours : alors, on remplace le code du dernier objet du tampon par le code correspondant à la somme des longueurs des segments, ceci est fait pour traiter le cas où les lignes discontinues sont celles d'un cercle ou d'une ellipse (dans ce cas les segments de droite servant à visualiser la conique peuvent être inférieurs à la longueur du plus long segment qui matérialise la ligne discontinue),
  - le point de fin du dernier segment du tampon est différent du point de début du segment de droite en cours : alors on ajoute dans le tampon, un segment de droite qui correspond à un segment de droite fictif dont les extrémités sont le point de fin du segment du tampon et le point de début du segment en cours. Si ce segment fictif est supérieur à la longueur fixée par l'utilisation, alors on affecte le code zéro au segment et il n'est pas enregistré dans le tampon. Sinon, on calcule le code du segment fictif comme pour un segment normal, puis on transforme ce code en une lettre avec les correspondances suivantes « A » pour 1, « B » pour 2, « C » pour 3, etc...

S'il est égal à zéro, deux situations peuvent se produire :

- soit le tampon est vide, dans ce cas il n'y a rien à faire,
- soit le tampon n'est pas vide, dans ce cas on analyse le tampon (fouille) pour voir si c'est une ligne discontinue, et déterminer le motif répétitif qui le compose, ensuite on vide le tampon.

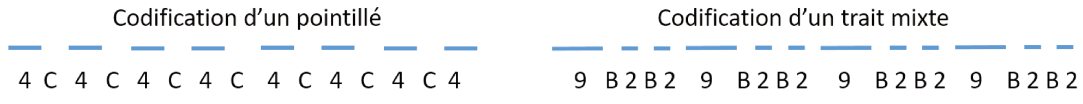


FIGURE 4.5 – Codification des lignes en trait discontinus

4.1.1.2 La fouille

La fouille dans le tampon, créé lors de la codification, consiste à rechercher le motif qui commence par un segment de droite réel (c'est à dire dans la codification par un chiffre) et qui se répète le plus souvent à l'intérieur de la chaîne du tampon sans se recouvrir, la longueur du motif est un multiple de 2.

Nous utilisons l'algorithme « SAP » (algorithme 2, page 59) que nous modifions pour l'adapter à la recherche des motifs répétitifs qui composent une ligne discontinue. Si nous prenons l'exemple de la figure 4.5 page 69 (codification des lignes en traits discontinus), l'objectif est d'identifier le motif « 4C », c'est le motif qui se répète le plus et qui commence par un segment de droite et dont la longueur est un multiple de 2.

Les modifications de l'algorithme général « SAP » sont les suivantes :
- seuls les paquets commençant par un nombre sont traités, il est inutile de traiter les paquets qui commencent par un segment de droite fictif.
- le tri par paquets s'arrête quand le nombre d'objets dans un paquet est un multiple de 2 et quand le nombre d'objets est supérieur ou égal à la longueur de la chaîne divisée par la profondeur du paquet moins 2.

L' algorithme « SAP » modifié pour traiter les lignes discontinues est nommé « SAP-DottedLine » (voir algorithme 4, page 70).

Algorithm 4 SAP-DottedLigne(R,p)

For $s \in R$ **do** $B[S[p]] := B[S[p]U\{s\}$ {Range les suffixes dans les paquets}
For $c \in N_+^*$ **do** {Pour chaque code numérique}
if $(|B[c]| \geq Seuil)$ **and** $(p \mod 2 = 0)$ **then** {Assez de suffixes ? }
  $Patternfound$ {et p multiple de 2 }
else {sinon }
  $SAP - DottedLigne(B[c], p+1)$ {appel récursif}

Dans notre premier exemple, celui de droite, nous avons la chaîne « 4C4C4C4C4C4C4C4 ». À la première itération nous avons 2 paquets. Le paquet des suffixes qui commencent par « 4 » et le paquet des suffixes qui commencent par « C ». Le paquet « C » est abandonné car il ne correspond pas à un chiffre (« C » codifie un espace, et, une ligne discontinue ne commence jamais par un espace). À la deuxième itération pour le paquet « 4 », nous avons uniquement des suffixes qui commencent par la lettre « C » et nous en avons 7. La longueur de la chaîne est égale à 15, la profondeur du paquet est égale à 2. L'algorithme est terminé car $7 > (15/2) - 2$. Le paquet « 4C » est le motif répétitif qui constitue la ligne discontinue.

Pour la deuxième codification de gauche « 9B2B2B9B2B2B9B2B2B9B2B2 », la longueur de la chaîne est 24. Le paquet « 9B » est présent 4 fois, mais 4 est inférieur au seuil 10 (10 est égal à (24/2) - 2), ce n'est pas le bon paquet. Idem pour le paquet « 2B » présent 8 fois. Le paquet « 9B2B » est présent 4 fois mais 4 n'est pas supérieur à 4 (4 est égal à (24/4)- 2). Idem pour les paquets « 2B2B » ou « 2B9B ». Par contre le paquet « 9B2B2B », de profondeur 6, est présent 4 fois, ce qui est supérieur à 2 (2 est égal à (24/6)-2). Le paquet

« 9B2B2B » est le motif répétitif qui constitue la ligne discontinue.

Remarque : on peut, par cette méthode, détecter des ensembles en lignes discontinues qui matérialisent des segments de droite, mais aussi, des cercles, des ellipses, des arcs de cercles ou des arcs d'ellipses. Ceci est dû à la notion de même angle et à la tolérance que l'on donne à cette notion. Si l'on donne une tolérance de 3 à 4° (même inférieure), les cercles ou les ellipses tracés avec des traits discontinus seront identifiés comme un ensemble dessiné en ligne discontinue.

Après l'identification d'un ensemble en traits discontinus, on vérifie si les points d'origine et de fin des segments de droites sont alignés. Si c'est le cas, c'est un segment de droite en trait discontinu, sinon, on vérifie à l'aide de l'équation d'un cercle (voir équation 4.1, page 77) ou d'une ellipse (voir équation 4.2, page 78).

4.1.1.3 L'interprétation et l'évaluation

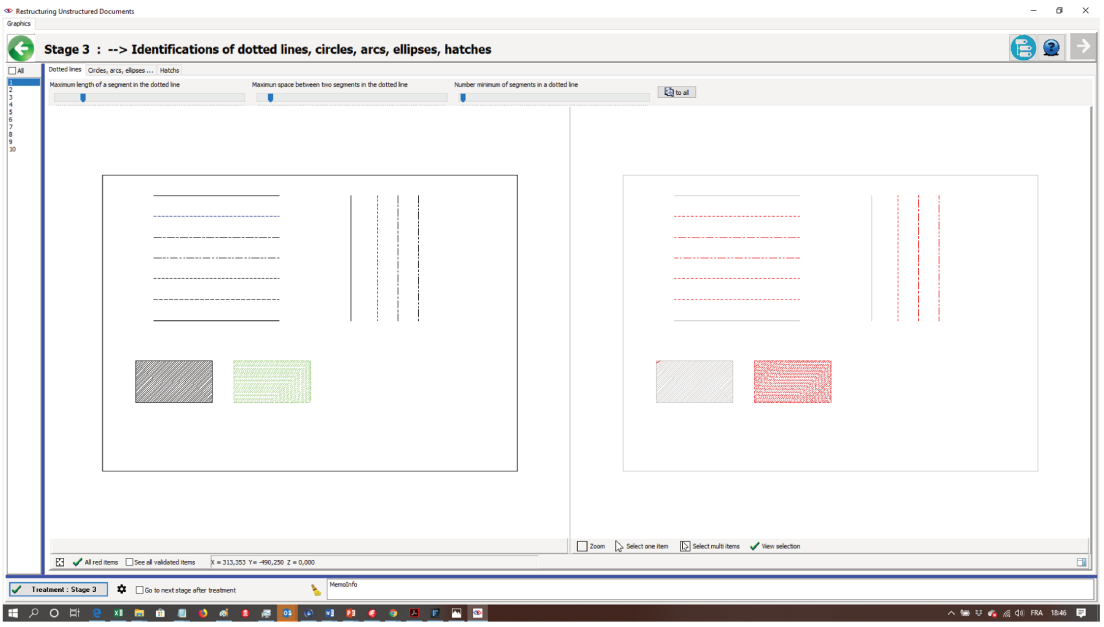


FIGURE 4.6 – Visualisation des lignes discontinues identifiées (en rouge)

La vue générale

Le principe est le même pour toutes les interprétations et évaluations graphiques. L'écran est divisé en 2 vues : à droite, nous avons l'image originale, et, à gauche, nous avons, tous les objets grisés (cette couleur est paramétrable) sauf ceux qui ont déjà été validés et qui sont en bleu (couleur également paramétrable) et ceux qui sont en cours de validation et qui sont en rouge (couleur paramétrable).

Les deux images sont synchronisées. À l'affichage on voit toute la vue, suivant le principe de « *zoom first* » et « *détails on demand* ». On peut zoomer avec la roulette ou définir une zone de zoom avec le bouton « *Zoom* » en bas

de la vue de droite (voir figure 4.7 le premier bouton à gauche) puis revenir au zoom total en cliquant sur le bouton correspondant à la vue totale (voir figure 4.8 le bouton de droite). Les ascenseurs sur les 2 vues permettent de déplacer l'image.

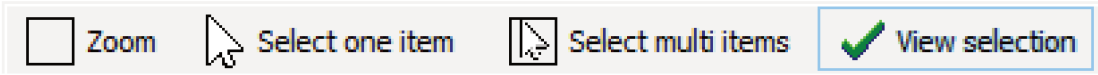


FIGURE 4.7 – Bouton en bas de la vue de droite pour la validation des objets

Dans l'exemple de figure 4.6, la vue de droite ne contient que des objets grisés ou des objets en rouge car aucun objet n'a été validé. Les objets déjà validés mais qui correspondent à un autre ensemble que les objets identifiés (dans le cas présent les lignes discontinues), ne s'affichent que si la case à cocher « *See all validated items*» est cochée (voir figure 4.8, case à cocher la plus à droite).

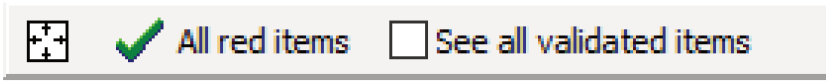


FIGURE 4.8 – Bouton en bas à gauche de l'écran pour la validation des objets

Tous les objets en rouge correspondent aux traits discontinus qui sont présents sur la partie gauche.

La validation globale

La validation de tous les objets en rouge se fait avec le bouton valider « *All red objects* » qui se trouve en bas gauche de l'écran (voir figure.4.8, le bouton du milieu).

Les objets s'afficheront en bleu une fois qu'ils auront été validés par l'utilisateur.

La validation partielle

Si tous les objets n'ont pas été correctement identifiés, il est possible de sélectionner un ou plusieurs objets présents sur la vue de droite pour modifier et changer leur affectation.

Pour faire cette sélection, deux outils sont disponibles :

— la sélection d'un seul objet se fait avec l'outil « *Select one item* », c'est l'outil du milieu sur la vue de droite (voir figure 4.7, le deuxième bouton en partant de la gauche). Pour sélectionner un objet, il suffit de sélectionner l'outil et de cliquer sur l'objet à sélectionner.

— la sélection de plusieurs objets se fait avec l'outil « *Select multi items* » à l'aide d'une fenêtre, c'est l'outil du milieu sur la vue de droite (voir figure 4.7, troisième bouton en partant de la gauche). Pour sélectionner plusieurs objets, il suffit de prendre l'outil et de dessiner un rectangle sur la vue de droite. Tous les objets compris dans ce rectangle seront sélectionnés.

En appuyant sur la touche « CTRL », il est possible de rajouter des objets en procédant comme ci-dessus. Remarque : si un ou plusieurs objets ont déjà été sélectionnés, ils seront dé-sélectionnés.

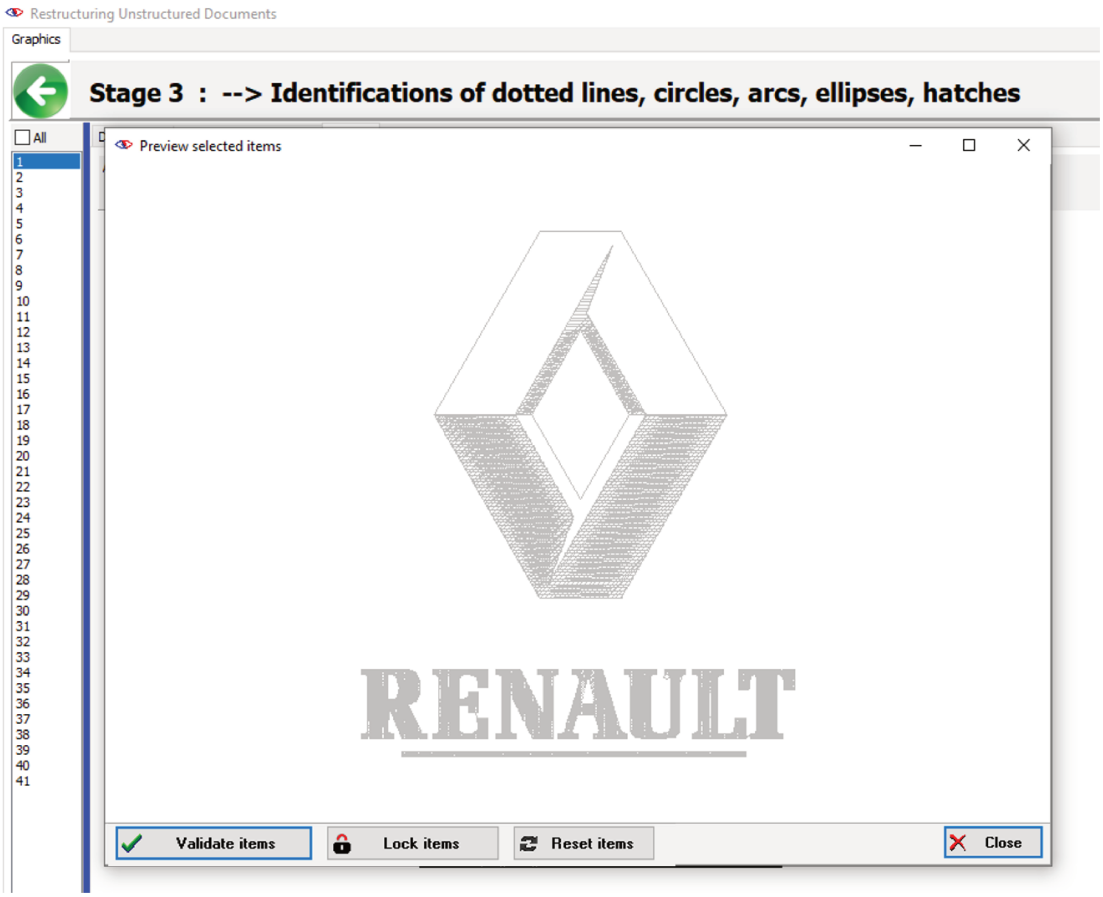


FIGURE 4.9 – Fenêtre pour la validation partielle de zone hachurée

Quand un ou plusieurs objets sont sélectionnés, le bouton situé à droite « *View selection* » devient actif (voir figure 4.7, page b72, bouton le plus à droite). L'appui sur ce bouton entraîne l'apparition d'une nouvelle fenêtre qui affiche l'ensemble des objets sélectionnés (voir figure 4.9, pour la validation de la zone hachurée, du logo Renault™). Trois actions sont alors possibles.

Ces trois actions correspondent aux trois boutons du bas de la fenêtre. Ils permettent :

— de valider l'ensemble des objets sélectionnés comme étant du type des objets à identifier (voir figure 4.10, bouton de droite « *Validate items* »),
— de supprimer l'affectation de tous les objets sélectionnés à un type d'objets (voir figure 4.10, bouton du milieu « *Lock items* »), et de bloquer toute réutilisation de ces objets sélectionnés lors des prochains traitements réalisés par l'ordinateur pour cette étape,
— de supprimer l'affectation de tous les objets sélectionnés au type d'objets traités (voir figure 4.10, bouton de gauche « *Reset items* »).

FIGURE 4.10 – Boutons de la fenêtre de validation

4.1.2 Étape 1-2 : Restructuration des hachures

La restructuration des hachures, ou des remplissages est une restructuration locale, c'est à dire qu'elle est limitée à un certains nombre d'objets graphiques qui sont des segments de droite et qui sont parallèles les uns aux autres. Bien sûr, les hachures sont présentes dans tout le fichier, mais la détection d'une hachure est locale.

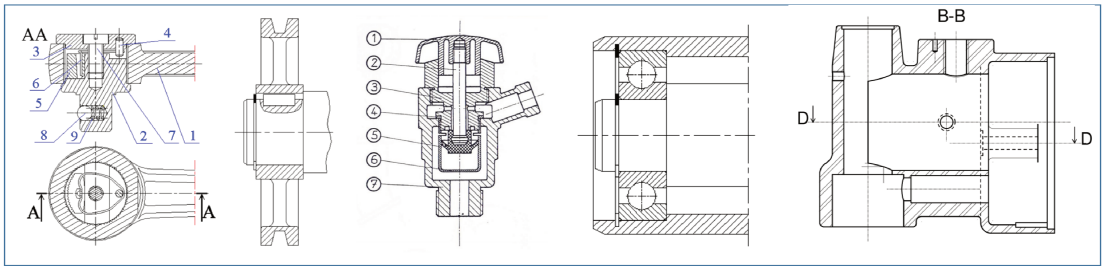


FIGURE 4.11 – Exemples de hachures en dessin industriel

4.1.2.1 La codification

Les paramètres

La première sous-tâche est une tâche utilisateur. Celui-ci fixe deux paramètres, avec une interface quasiment identique à celle décrite pour les lignes en traits discontinus :

— le premier paramètre est la distance maximale entre deux segments de droite qui composent la hachure. La valeur par défaut, prise par le système, est de 5 millimètres. L'utilisateur peut changer cette valeur (elle sera proposée lors du prochain traitement).

— le deuxième paramètre est le nombre de segments parallèles nécessaire pour considérer que les segments de droite forment une hachure. La valeur par défaut, prise par le système, est de 4. L'utilisateur peut changer cette valeur (elle sera proposée lors du prochain traitement).

L'utilisateur fixe ces paramètres avec un curseur. Sur l'écran, comme pour les lignes en traits discontinus, en superposition de l'image graphique, s'affiche un carré dont la taille, choisie par l'utilisateur, correspond à la distance maximale entre deux segments. De cette façon l'information de distance est visualisée et peut-être comparée avec la distance entre les hachures (c'est le petit point rouge dans la vue de gauche de la figure 4.12, page 75).

Le tampon local

La deuxième sous-tâche est une tâche système qui concerne la codification. Tout objet qui n'est pas un segment de droite reçoit le code 0. Tout segment de droite qui n'est pas précédé d'un segment de droite parallèle et dont la distance entre ce segment et le segment précédent est inférieure à la distance fixée par l'utilisateur reçoit le code 0.

Pour chaque objet graphique du fichier d'entrée, deux cas peuvent se produire :

— s'il est différent de zéro : on l'enregistre dans un tampon,
— s'il est égal à zéro, deux situations peuvent se produire :
  — soit le tampon est vide, dans ce cas il n'y a rien à faire,
  — soit le tampon n'est pas vide, dans ce cas il est analysé (fouille) pour identifier les hachures, puis vidé.

4.1.2.2 La fouille

La fouille dans le tampon est très simple : il suffit de contrôler que le nombre d'objets dans le tampon est supérieur ou égal au nombre d'éléments minimum fixé par l'utilisateur. Si c'est le cas, l'ensemble des objets forme une hachure.

Une fois qu'une hachure est identifiée, on la compare à la hachure précédente si elle existe. La comparaison consiste à regarder si les deux boîtes englobantes des deux lignes de hachures consécutives se recouvrent avec une tolérance égale au dixième de la somme des surfaces des deux boîtes englobantes. Si c'est le cas, on fusionne les deux ensembles d'objets pour former une seule hachure.

4.1.2.3 L'interprétation et l'évaluation

L'interprétation et l'évaluation se fait comme pour les lignes discontinues, à la seule différence que l'on traite des hachures et non des lignes discontinues. La figure 4.11 montre le résultat (vue de droite, avec les segments en rouge) de l'identification des hachures.

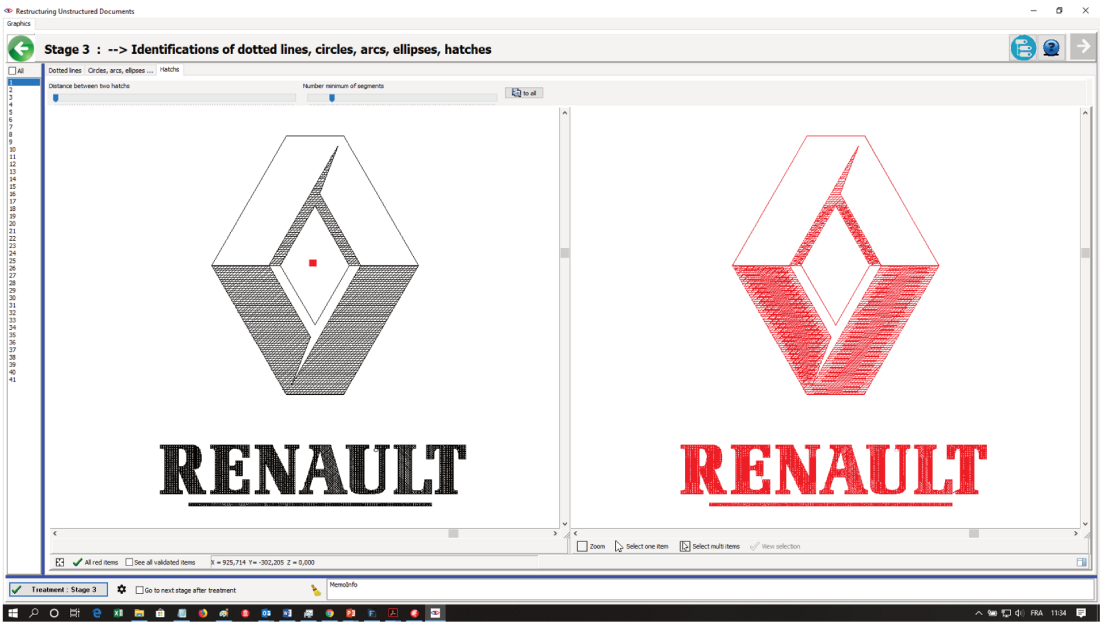


FIGURE 4.12 – Exemples d'identification des hachures du logo Renault™

4.1.3 Étape 1-3 : Restructuration des coniques

Les coniques que nous traitons sont les cercles, les ellipses, les arcs de cercles et les arcs d'ellipses. La restructuration des coniques est une restructuration locale, c'est à dire qu'elle est limitée à un certains nombre d'objets graphiques qui sont des segments de droite et qui sont contiguës. Bien sûr, les coniques sont présentes dans tout le fichier, mais la détection d'une conique est locale.

4.1.3.1 La codification

Les paramètres

La première sous-tâche est une tâche utilisateur. L'utilisateur fixe trois paramètres, avec une interface quasiment identique à celle décrite pour les lignes en traits discontinus :

— la longueur du segment de droite le plus long qui peut entrer dans la composition d'une conique. La valeur par défaut, prise par le système, est de dix millimètres, l'utilisateur peut changer cette valeur (elle sera proposée lors du prochain traitement).
— l'angle maximum qui existe entre deux segments consécutifs. La valeur par défaut, prise par le système, est de cinq degrés, l'utilisateur peut changer cette valeur (elle sera proposée lors du prochain traitement).
— le nombre de segments consécutifs minimum. La valeur par défaut, prise par le système, est de quinze, l'utilisateur peut changer cette valeur (elle sera proposée lors du prochain traitement).

Le tampon local

La deuxième sous-tâche est une tâche système qui codifie tous les objets du fichier d'entrée. Toutes les polylignes sont décomposées en segments de droite. Les cercles ou les arcs de cercles peuvent être présents dans les polylignes, c'est pour cette raison que l'on décompose une polyligne en segments de droite. Après cette décomposition, tout objet qui n'est pas un segment de droite reçoit le code 0. Tout segment de droite qui est supérieur à la longueur fixée par l'utilisateur reçoit le code 0. Tout segment de droite qui n'est pas suivi d'un segment de droite dont le point de début est le même que le point de fin du segment de droite reçoit le code 0.

Tout segment de droite qui n'est pas suivi d'un segment de droite de même angle en tenant compte de la tolérance fixée par l'utilisateur (deuxième paramètre) reçoit le code 0. Tout segment de droite qui n'est pas suivi d'un segment de droite de même longueur (compris entre la moitié et le double de la longueur du segment courant) reçoit le code 0.

Remarque : dans certains cas, le point de début et de fin du segment de droite ont été permutés, ce qui oblige, en réalité, à faire le test pour les deux extrémités du segment de droite. C'est le cas des logiciels qui permettent de masquer certains objets. Pour ces logiciels tous les segments de droite verticaux sont dessinés de haut en bas et tous les segments horizontaux sont dessinés de la gauche vers la droite.

Pour chaque objet graphique du fichier d'entrée, deux cas peuvent se produire :

- s'il est différent de zéro : on calcule son code en tronquant la longueur du segment de droite multiplié par dix et divisé par la longueur du segment maximum, et en ajoutant 1 au résultat, puis on l'enregistre dans un tampon,
- s'il est égal à zéro, deux situations peuvent se produire :
  - soit le tampon est vide, dans ce cas il n'y a rien à faire,
  - soit le tampon n'est pas vide, on analyse le tampon (fouille) pour voir si c'est une conique, et ensuite on vide le tampon.

4.1.3.2 La fouille

La fouille dans le tampon est simple : il suffit de contrôler que, pour chaque segment contenu dans le tampon, le point de fin est bien sur une conique. En premier, on regarde si la forme trouvée est fermée (les coordonnées du premier point correspondent aux coordonnées du dernier point), puis on procède en deux temps :

Dans un premier temps, on va regarder si les points sont sur un cercle. Trois points sont choisis. Si la forme n'est pas fermée, c'est le premier point, le dernier point et le point du milieu. Si la forme est fermée c'est le premier point, puis on divise le nombre de points par trois, et on prend le point qui correspond au tiers, puis le point qui correspond aux deux tiers du nombre de points de la forme (les trois points ne doivent pas être alignés, s'ils le sont, ce n'est pas une conique). Ensuite, on calcule l'équation du cercle passant par trois points. Puis on vérifie que tous les points de fin de chaque segment de droite sont bien sur le cercle avec une certaine tolérance (égale au paramètre longueur divisé par cinq). Si c'est le cas, l'équation du cercle est trouvée. Sinon on va regarder si c'est une ellipse. Si c'est un cercle et si la forme est ouverte, c'est un arc de cercle pour lequel on calcule l'angle de départ du premier point par rapport à son centre et à l'horizontale, puis sur le même principe l'angle du dernier point par rapport au centre et à l'horizontale.

Pour rappel : l'équation d'un cercle de centre $x_c = a$, $y_c = b$ et de $rayon = r$ est :

$$(x - a)^2 + (y - b)^2 = r^2 \tag{4.1}$$

Nous avons trois inconnues (a, b, r) donc la nécessité d'avoir trois équations qui sont obtenues avec les trois points non alignés.

Dans un deuxième temps, si les points ne sont pas sur un cercle, on va regarder s'ils sont sur une ellipse. On procède comme pour un cercle mais on prend cinq points pour déterminer l'équation de l'ellipse. Ces cinq points sont répartis uniformément et non alignés sur les points du tampon (s'ils sont alignés ce n'est pas une ellipse). Avec ces cinq points, l'équation de la conique est calculée, puis on vérifie que tous les points de fin de chaque segment de droite sont bien sur la conique avec une certaine tolérance (égale au paramètre longueur divisé par

cinq). Si la forme est ouverte, c'est un arc d'ellipse pour lequel on calcule l'angle de départ du premier point par rapport au centre de l'ellipse et à l'horizontale, puis, sur le même principe, l'angle du dernier point par rapport au centre de l'ellipse et à l'horizontale.

Remarque : on applique le même type de calcul de la conique pour les ensembles de traits discontinus qui peuvent correspondre à une conique.

Pour rappel : l'équation d'une ellipse de premier foyer x_{c1}, y_{c1}, de deuxième foyer x_{c2}, y_{c2} et de corde $corde = r$ est :

$$(x - (x_{c1})^2 + (y - y_{c1})^2 + (x - x_{c2})^2 + (y - y_{c2})^2 = r^2 \tag{4.2}$$

Nous avons cinq inconnues $(x_{c1}, y_{c1}, x_{c2}, y_{c2}, r)$ donc la nécessité d'avoir 5 équations avec les cinq points (non alignés trois par trois).

4.1.3.3 L'interprétation et l'évaluation

L'interprétation et l'évaluation se fait comme pour les lignes discontinues, à la seule différence que l'on traite des cercles et des ellipses.

4.1.4 Étape 1-4 : Restructuration des formes simples

La restructuration des formes simples se déroule exactement de la même façon que la restructuration des coniques. Il suffit de changer les paramètres avec les curseurs et ce sont les formes simples (triangles, carrés, rectangles, losanges, parallélogrammes) qui peuvent être identifiées.

4.2 Étape 2 : Restructuration des objets textuels

La restructuration des objets textuels (voir figure 4.13) en unités lexicales consiste à :

— restructurer les caractères (si nécessaire),
— restructurer les mots,
— restructurer les phrases (ensembles de mots),
— restructurer les paragraphes (peu utile dans le domaine des schémas).

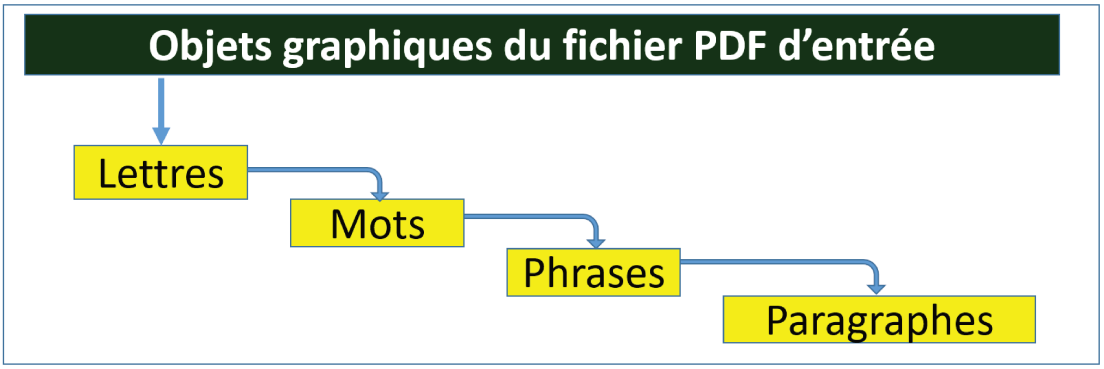


FIGURE 4.13 – Restructuration des unités lexicales

Nous avons repris le schéma (voir figure 4.14) qui explique bien les différentes unités lexicales. Cette figure est extraite de « XCDF : Un format canonique pour la représentation de documents » dont les auteurs sont : Jean-Luc Bloechle, Maurizio Rigamonti, Denis Lalanne et Rolf Ingold. A gauche, dans la première colonne, sont représentées des lettres ou des assemblages de lettres, dans la deuxième colonne apparaissent les mots et les espaces entre les mots, dans la troisième colonne ce sont les phrases, et dans la quatrième colonne les paragraphes, ceci avec les termes que nous utilisons pour les unités lexicales dans le cadre des plans et schémas.

FIGURE 4.14 – Exemple de restructuration des mots et paragraphes : fusion des primitives textuelles extrait de « XCDF : Un format canonique pour la représentation de documents » Jean-Luc Bloechle, Maurizio Rigamonti, Denis Lalanne, Rolf Ingold

4.2.1 Les différentes formes des textes

Avant de décrire ces restructurations, il est important de bien définir les différentes formes qui peuvent être présentes dans les fichiers PDF pour matérialiser les textes.

4.2.1.1 Les trois principales formes des textes

La représentation des objets graphiques textuels à l'intérieur d'un fichier PDF prennent trois formes principales :

L'objet textuel est une lettre, au sens informatique du terme, c'est à dire que l'on connaît la police du texte, son nom, sa taille, son orientation, son style et l'on connaît aussi les codes des lettres. Ces codes identifient de façon unique la lettre dans la codification UTF-8 (initialement développée par l'ISO et la CEI dans la norme internationale ISO/CEI 10646). Cette codification normalisée est la plus répandue dans le monde informatique. Elle intègre les codifications ASCII et UNICODE. Nous appelons cette forme la forme principale « lettre ».

À l'opposé de cette représentation, on a le cas où l'objet graphique textuel n'a aucune information sur le nom de la police, sa taille, son orientation, son style ni son code. On a uniquement le glyphe ou les glyphes qui sont les représentations graphiques des lettres. Ces glyphes sont matérialisés par un objet graphique vectoriel composé d'une ou plusieurs polylignes, en général, remplies. Nous appelons cette forme la forme principale « glyphe ».

Et, entre ces deux extrêmes, nous avons le cas des objets graphiques textuels pour lesquels nous avons, comme dans la forme principale « lettre », toutes les informations sur la police, son nom, sa taille, son orientation, son style, mais le code de chaque lettre à été remplacé par un code décimal qui commence à 1. Par exemple, tous les « E » sont remplacés par le code 1, les « L » par le code 2, les « C » par le code 3. Avec cette codification, le texte « ELEC » est codé par « 1213 ». Nous appelons cette forme la forme principale « intermédiaire ».

Il existe d'autres formes de représentation des textes, mais elles peuvent être ramenées à l'une des 3 formes principales mentionnées ci-dessus.

4.2.1.2 Détermination de la forme principale

Cette sous-tâche est une tâche système. Elle détermine la forme principale des objets graphiques textuels. Elle analyse tous les objets graphiques du fichier d'entrée.

Si des objets graphiques matérialisent des lettres (police + code), deux cas peuvent se produire :

— soit il existe des codes inférieurs ou égaux à la valeur décimale 31, alors nous avons la forme principale « intermédiaire ». Dans certains cas, il peut exister, dans le fichier d'entrée, une table de correspondance entre le code présent dans la lettre et le code du caractère qu'il représente. Si

c'est le cas, cela nous conduit à la forme principale « lettre »,
— soit, tous les codes sont supérieurs ou égaux au code décimal 32 (qui correspond au caractère espace), alors nous avons la forme principale « lettre ».

S'il n'existe aucun objet graphique qui matérialise des lettres, comme mentionné ci-dessus, deux cas peuvent se produire dans les fichiers d'entrée de type PDF :

— il existe des objets graphiques qui sont composés d'une ou plusieurs polylignes, alors nous avons une très forte probabilité que ces formes correspondent aux glyphes des lettres. Nous avons dans ce cas, la forme principale « glyphe ». Cette probabilité peut s'affiner en tenant compte de la position, de l'écartement et de la taille de ces glyphes. Si cette probabilité est forte, on considère que ces glyphes sont la représentation de lettres. Dans le cas contraire, nous n'avons pas de lettres dans le fichier d'entrée (c'est un cas très rare).
— il n'existe pas d'objet graphique de ce type dans le fichier d'entrée. Dans ce cas, nous avons une très forte probabilité qu'il n'existe pas de texte dans ce fichier, des textes pourraient exister sous forme purement vectorielle (par exemple segment de droite), mais cette hypothèse n'est pas traitée dans le cadre de ces travaux de recherche. Les lettres seront traitées comme des objets graphiques vectoriels et non pas comme des lettres.

4.2.2 Étape 2-1 : Restructuration des lettres

La restructuration des lettres est différente suivant la forme principale du texte que nous venons de décrire.

La sous-tâche suivante est une tâche utilisateur. Elle n'existe que si la forme principale est la forme principale « glyphe ». L'utilisateur fixe un paramètre. Ce paramètre représente la valeur maximale du coefficient de divergence entre deux glyphes. Si les deux glyphes sont exactement les mêmes, le coefficient de divergence est égal à zéro. Plus les glyphes sont différents, plus le coefficient de divergence est important (voir le calcul du coefficient de divergence deux glyphes, paragraphe : 4.2.2.2, page 82). L'utilisateur détermine également un ou plusieurs noms des polices de caractère qui sont présentes dans le document. Ceci se fait avec une liste déroulante des noms des polices du système d'exploitation.

Les sous-tâches suivantes de la restructuration des caractères sont des tâches systèmes. Elles sont différentes pour chacune des formes principales des objets graphiques textuels.

4.2.2.1 Restructuration de la forme principale « lettre »

Dans le cas de la forme principale « lettre », nous n'avons rien à faire. Toutes les informations concernant la lettre, sa police et son code sont présentes dans l'objet graphique textuel du fichier PDF d'entrée.

4.2.2.2 Restructuration de la forme « intermédiaire »

Les sous-étapes de la restructuration

Pour la forme principale « intermédiaire », nous avons un code et nous devons trouver la lettre qui correspond à ce code dans la police dont on connaît le nom, la taille, l'orientation et le style. Pour trouver cette correspondance nous procédons en quatre sous-étapes et nous générons deux fichiers intermédiaires dans une sous-étape préparatoire.

Le premier fichier contient les objets graphiques textuels (avec les codes des lettres qui commencent par le code 1). Le deuxième fichier est le même fichier dans lequel nous avons remplacé chaque objet textuel par son glyphe. Par comparaison des deux fichiers, nous pouvons associer à chaque code son glyphe.

La première sous-étape, des quatre sous-étapes, regroupe, dans un cluster, tous les objets textuels qui ont le même code. Ces clusters sont appelés « cluster-code ». À l'intérieur de ces clusters, les objets textuels qui ont le même nom de police, la même taille, la même orientation, le même style sont regroupés pour former les clusters dénommés « cluster-code-groupe ».

La première sous-étape, des quatre sous-étapes, regroupe tous les objets textuels.

La deuxième sous-étape recherche la taille de la police de référence du groupe. Cette taille correspond à la somme de tous les objets graphiques divisée par le nombre d'objets graphiques du groupe. Cette opération est répétée pour chaque groupe défini dans la première sous-étape.

La troisième sous-étape recherche, pour chaque code dans un groupe (« cluster-code-groupe »), l'objet graphique dont la taille est la plus proche de la taille de la police de référence du groupe. Cet objet graphique devient l'objet graphique de référence du code (« cluster-code »). Cette opération est répétée pour chaque groupe défini dans la première sous-étape et pour chaque code dans ce groupe.

La quatrième sous-étape compare chaque objet graphique de référence du (« cluster-code »), avec tous les caractères de la police de référence du groupe. Cette comparaison donne un coefficient de divergence entre les deux glyphes comparés (voir le paragraphe : C.1 pour la méthode de calcul). Les cinq caractères de la police de référence, dont les coefficients de divergence sont les plus faibles, sont conservés et ordonnés, du moins divergent au plus divergent. Cette opération est répétée pour chaque code.

La méthode de comparaison des glyphes

Le coefficient de divergence d'un objet graphique avec l'un des caractères de la police de référence se calcule de la façon suivante :

— le coefficient de divergence est égal à la valeur absolue de la différence du nombre de polylignes de l'objet et du caractère, différence multipliée par 256,

— on ajoute à ce coefficient de divergence la valeur absolue de la différence du nombre de points des polylignes de l'objet et du caractère, différence

multipliée par 16 puis divisée par le cumul des points des polylignes de l'objet et du caractère,
— puis, on ajoute à ce coefficient de divergence la valeur absolue de la différence entre la somme des segments des polylignes de l'objet graphique, différence divisée par la longueur de la diagonale de la boîte englobante et la somme des segments des polylignes du caractère divisé par la longueur de la diagonale de sa boîte englobante.

Le coefficient de divergence est un réel. Il est calculé pour un objet graphique de référence avec tous les caractères de la police de référence. Pour des raisons d'optimisation, il n'est pas nécessaire de faire tous ces calculs. On peut considérer qu'un objet ne ressemble pas à un caractère si son coefficient de divergence est supérieur à 256.

Remarque : le coefficient de divergence est presque insensible à la taille des glyphes.

4.2.2.3 Restructuration de la forme principale « glyphe »

La restructuration de la forme principale « glyphe » est la plus complexe. Elle se fait en cinq sous-étapes.

La première sous-étape détermine l'orientation de chaque objet graphique textuel. Cette orientation est déterminée à l'aide du caractère précédent et/ou du caractère suivant, quand ils existent. Cette orientation est indéterminée si nous n'avons pas de caractère précédent et/ou suivant.

Un objet graphique textuel est l'objet graphique textuel précédent (ou suivant) l'objet courant, si cet objet graphique textuel est juste l'objet précédent (ou suivant) dans l'ordre chronologique des objets, et si, la distance entre les coordonnées X et Y des deux objets est :
— inférieure à la somme des largeurs des boîtes englobantes des deux objets graphiques quand les deux objets sont alignés horizontalement : même valeur X (voir note [1])
— inférieure à la somme des hauteurs des boîtes englobantes quand les deux objets sont alignés verticalement : même valeur Y (voir note 1)
— inférieure à la moitié de la diagonale des boîtes englobantes dans les autres cas.

L'orientation de l'objet graphique textuel courant est calculé à partir de ses coordonnées X et Y, et des coordonnées X et Y de l'objet graphique textuel précédent et/ou suivant.

Quatre orientations de texte sont définies :
— les textes sont horizontaux (Angle 0°) : les deux objets graphiques ont le même X (voir note 1)
— les textes sont verticaux (angle à 90°) : les deux objets graphiques ont le même Y (voir note 1)
— les textes sont inclinés à 45° (angle à 45°) : les deux objets graphiques

1. Le calcul de même X ou Y se fait avec une tolérance égale au millimètre.

sont sur une ligne à 45°(voir note [2])
— l'angle des textes est indéterminé, pour deux raisons :
  — soit il n'existe pas d'objet graphique précédent ou suivant comme défini ci dessus,
  — soit les deux objets graphiques sont sur une ligne dont l'angle n'est pas à 45°.

Remarque : les notions de même X ou Y et d'angle à 45° se font avec une certaine tolérance.

La deuxième sous-étape établit, pour chacune des quatre orientations, une clusterisation des éléments graphiques textuels. L'utilisateur a fixé, avec un curseur, le coefficient de divergence maximum qui sert à déterminer l'appartenance d'un objet graphique à un cluster.

La clusterisation des éléments graphiques utilise la méthode DBSCAN *Density-Based Spatial Clustering of Applications with Noise* [Est+96] que nous adaptons à notre problème. Le coefficient de divergence entre deux glyphes (voir paragraphe : 4.2.2.2) est calculé entre tous les objets graphiques textuels. Dès que deux objets graphiques ont un coefficient de divergence inférieur à celui défini par l'utilisateur, ces objets graphiques sont rangés dans le même cluster. Quatre cas peuvent se produire :
— les deux objets n'appartiennent à aucun cluster, alors un nouveau cluster est formé avec ces deux objets,
— un seul des objets appartient à un cluster, alors l'autre objet est rajouté à ce cluster,
— les deux objets appartiennent à des clusters différents, alors les deux clusters sont fusionnés
— les deux objets appartiennent déjà au même cluster, alors rien n'est à faire.

La troisième sous-étape choisit un représentant du cluster. L'objet graphique qui a la plus haute boîte englobante (en tenant compte de l'orientation de l'objet graphique que l'on a attribuée lors de la première sous-étape) est pris comme objet graphique de référence du cluster.

La quatrième sous-étape détermine trois tailles de police de caractères de référence pour l'ensemble des objets graphiques. La taille moyenne d'un dixième des plus grands objets graphiques textuels est calculée en tenant compte là aussi de l'orientation. Cette taille moyenne donne la taille de la première police de référence (c'est la plus grande). Les deux autres tailles sont obtenues en prenant 1) les deux tiers de la première taille 2) le tiers de la première taille.

La cinquième sous-étape compare chaque représentant du cluster (défini à la sous-étape trois), avec les polices de caractères dont les noms ont été donnés par l'utilisateur et dont les tailles ont été déterminées à la sous-étape quatre. Cette comparaison tient compte de l'orientation des objets graphiques du cluster. Comme pour la restructuration de la forme principale « intermédiaire », et en appliquant le même principe, une liste des cinq caractères les plus représentatifs est associée au cluster.

2. le calcul d'égalité d'angle se fait à 5° près.

4.2.2.4 L'interprétation et l'évaluation

L'interprétation et l'évaluation sont différentes de celles des éléments graphiques.

La figure 4.15 affiche le résultat de l'identification des caractères. Dans l'exemple affiché, les objets graphiques textuels sont de la forme principale « intermédiaire ». L'écran se divise en 2 grandes parties, la partie de droite qui est composée de trois bandes et la partie gauche sur un fond clair.

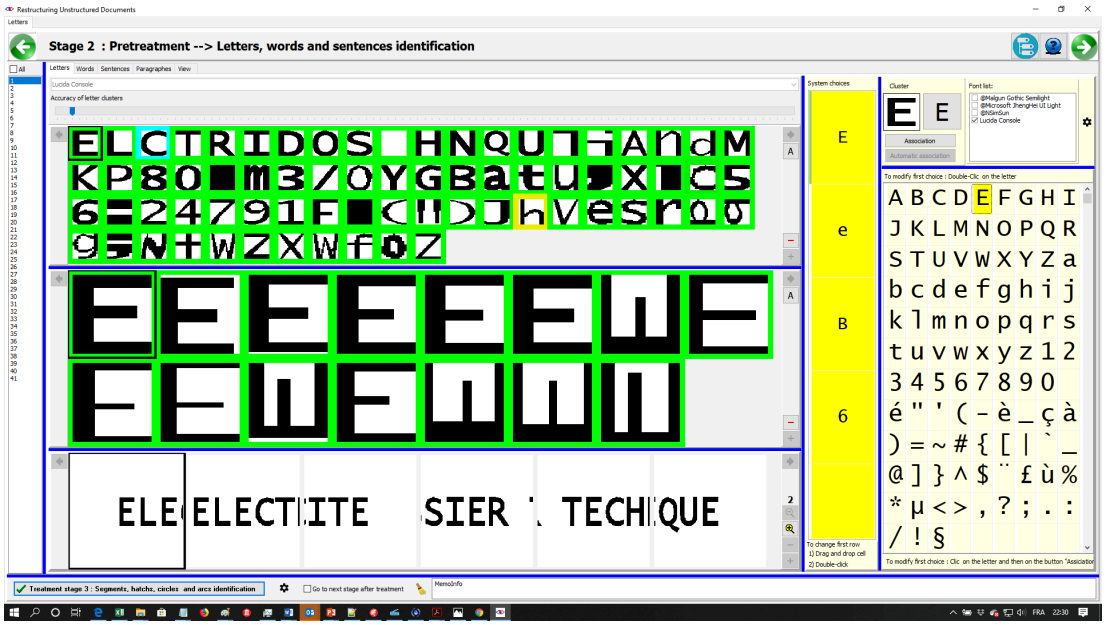


FIGURE 4.15 – Écran de validation de l'identification des caractères

La partie droite visualise les caractères du dossier sur 3 bandes différentes.

La première bande affiche tous les caractères des objets textuels de référence associés à chaque « cluster-code » (dans l'exemple, 71 codes qui ont créé 71 « cluster-code » différents). L'affichage des codes se fait dans l'ordre chronologique d'apparition dans le fichier PDF. Les caractères, entourés de vert, sont identifiés par le système avec une très grande fiabilité. Les caractères, entourés de bleu, sont les caractères pour lesquels le système a des doutes importants. Par exemple, pour la lettre C, le glyphe peut correspondre au C majuscule ou c minuscule. Le système lève cette incertitude en tenant compte des tailles des caractères, mais nous attirons l'attention de l'utilisateur sur la possibilité d'une mauvaise décision. Pour les caractères entourés de jaune, le doute est moins fort, mais nous souhaitons que l'utilisateur vérifie le choix que propose le système. Un clic sur l'un des caractères affiche les informations associées à ce « cluster-code » dans la deuxième bande.

La deuxième bande affiche, pour le « cluster-code » sélectionné, les lettres des objets graphiques de référence de ce que nous avons appelé « cluster-code-groupe ». Ces lettres correspondent à des tailles et des orientations différentes. Un clic sur l'un des caractères affiche les informations associées à ce « cluster-code-groupe » dans la troisième bande.

La troisième bande affiche, pour le « cluster-code-groupe » sélectionné, les

différents objets graphiques textuels qui appartiennent à ce cluster. Ces objets graphiques textuels s'affichent dans leurs environnements. Ceci permet à l'utilisateur de bien voir la lettre dans tous ses cas d'utilisation.

Tout un ensemble de boutons sur ces bandes permettent d'avoir plus ou moins d'informations et, de cette façon, de pouvoir grossir et faire défiler ces informations pour faciliter la décision de l'utilisateur.

La partie gauche sur un fond clair permet de modifier le choix fait par le système.

Sur le premier tableau, sur fond jaune, sont affichés, en colonne, les caractères que le système a trouvés comme caractères approchants en les classant verticalement (du haut vers le bas) du plus approchant au plus différent. L'utilisateur peut glisser et déposer l'un de ces caractères en première position. Cette action modifie le choix du système. Il peut également double cliquer sur un caractère, ce qui a le même effet que le glisser déposer. Dans le cas où le système n'aurait pas identifié le caractère parmi les meilleurs choix, l'utilisateur peut, grâce au tableau de l'ensemble des lettres qui se trouve à droite du tableau jaune, double cliquer sur la lettre de son choix, ce qui a pour effet de modifier le choix du système en plaçant en première position ce caractère, ou bien cliquer sur la lettre, elle s'affiche à côté de l'image du cluster, puis cliquer sur le bouton « Associate » afin que la lettre remplace le premier choix fait par le système. Nous verrons que l'utilisateur peut modifier la lettre associée à un glyphe lors de la restructuration des mots, des phrases et des paragraphes.

4.2.3 Étape 2-2 : restructuration des mots

La restructuration des mots est l'étape incrémentale qui suit immédiatement la restructuration des caractères. Cette étape incrémentale va lier les objets graphiques textuels les uns avec les autres pour former des unités lexicales qui, dans le cas présent, sont les mots (voir figure 4.14, page 79, deuxième colonne). Nous verrons aussi que, dans cette étape, l'utilisateur peut modifier le caractère affecté à un glyphe.

4.2.3.1 La codification

Les paramètres

La première sous-tâche est une tâche utilisateur. L'utilisateur fixe un paramètre, avec une interface quasiment identique à toutes celles décrites pour fixer un paramètre pour les lignes en traits discontinus. Ce paramètre fixe la distance maximum entre 2 lettres. C'est à dire l'espace maximum qui peut exister entre les boîtes englobantes de 2 caractères consécutifs. Ce paramètre fixe la largeur maximale du rectangle bleu de la figure qui matérialise la distance entre deux boîtes englobantes (voir figure 4.16, page 87). Les boîtes englobantes des lettres sont les rectangles rouges qui entourent les lettres.

Le tampon local

La deuxième sous-tâche est une tâche système qui traite directement le résultat de l'étape précédente. Au début, le tampon est rempli avec la première lettre identifiée lors de l'étape précédente, ensuite, deux cas peuvent se produire.

Soit la boîte englobante du caractère courant est à une distance inférieure à celle fixée, par l'utilisateur, dans ce cas, la lettre est mise dans le tampon. Soit la distance est supérieure, alors le tampon est passé à l'étape fouille, puis il est vidé et la lettre suivante est traitée.

FIGURE 4.16 – Visualisation des boîtes englobantes des lettres (rouge) et de l'espace entre les boîtes (bleu)

4.2.3.2 La fouille

La fouille dans le tampon consiste à assembler les lettres les unes avec les autres et créer ainsi une entité de plus haut niveau qui est le mot. Cette entité de plus haut niveau conserve les liens vers toutes les lettres de l'étape précédente qui ont servi à créer ce mot.

4.2.3.3 L'interprétation et l'évaluation

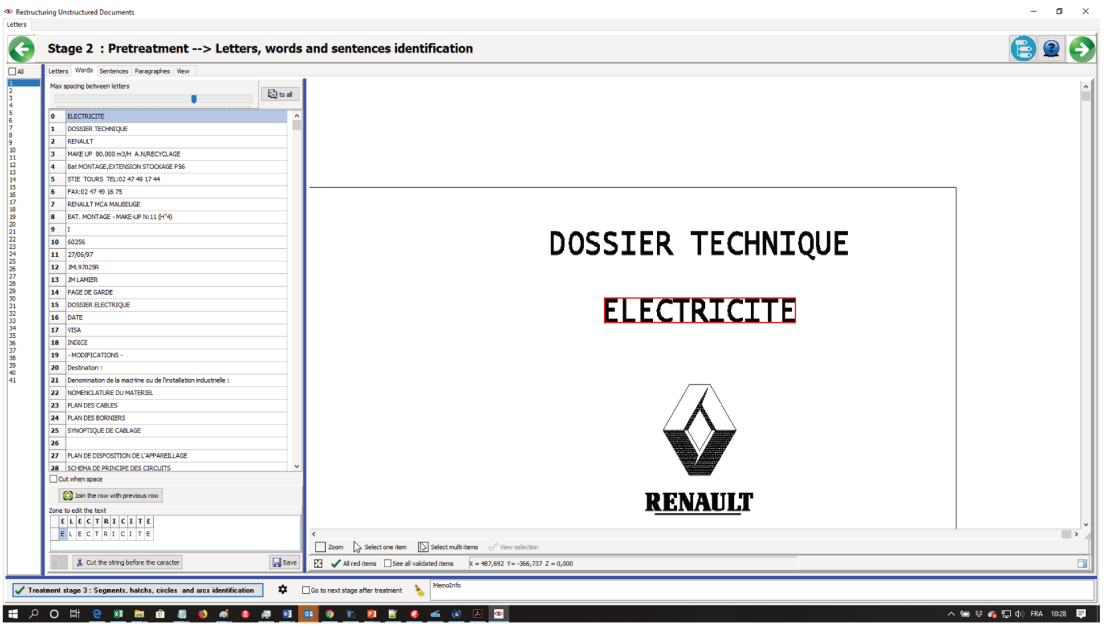


FIGURE 4.17 – Exemple de visualisation et de modification des mots

L'interprétation et l'évaluation sont différentes de celles des éléments graphiques et de la restructuration des lettres.

La figure 4.17, page 87 affiche le résultat de l'identification des caractères. L'écran se divise en 2 grandes parties, la partie de gauche où est affichée l'ensemble des mots reconstitués, et la partie droite qui visualise le mot dans le folio quand l'utilisateur clique sur un mot. Il est entouré d'un cadre rouge qui visualise la boîte englobante du mot.

La case à cocher « *Cut when space* » (voir figure 4.18, page 88) permet d'arrêter la fusion des lettres pour former un mot quand un caractère « espace » est rencontré, ceci quand elle est cochée. Dans le cas contraire l'association des lettres tient compte uniquement de l'espace fixé par l'utilisateur.

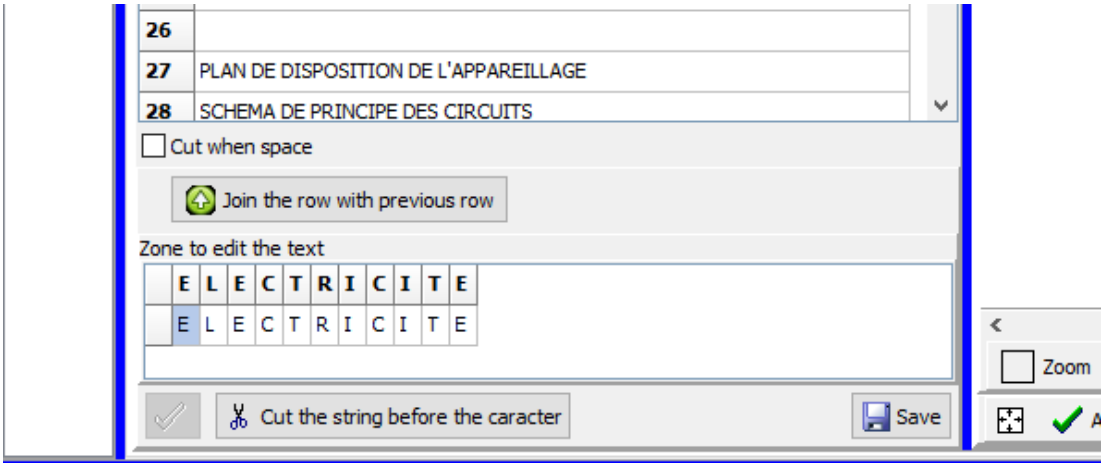


FIGURE 4.18 – Interface de modification des mots

Avec l'interface de modification des mots, l'utilisateur peut :
— fusionner 2 mots, bouton « *Join the row with previous row* »,
— couper un mot en deux, bouton « *Cut the string before the caracter* »,
— modifier les lettres du mot à l'aide de la zone « *Zone to edit the text* » si une lettre ou plusieurs lettres sont modifiées et que l'utilisateur valide ce choix, ces modifications s'appliquent à toutes les lettres associées aux glyphes des lettres modifiées.
— sauver l'ensemble des opérations réalisées (fusion, coupure, modification) avec le bouton « *Save* ».

4.2.4 Étape 2-3 : Restructuration des phrases

La restructuration des phrases est l'étape incrémentale qui suit immédiatement la restructuration des mots. Cette étape incrémentale va lier les objets graphiques textuels (les mots), les uns avec les autres pour former des unités lexicales qui, dans le cas présent, sont les phrases (voir figure 4.14, page 79, troisième colonne). Nous verrons aussi que, dans cette étape, l'utilisateur peut encore modifier le caractère affecté à un glyphe.

4.2.4.1 La codification

Les paramètres

La première sous-tâche est une tâche utilisateur. Ce dernier fixe un paramètre, avec une interface quasiment identique à toute celles décrites pour fixer un paramètre pour les lignes en traits discontinus. Ce paramètre fixe la distance maximum entre 2 mots. C'est à dire l'espace maximum qui peut exister entre les boîtes englobantes de 2 mots consécutifs. Ce paramètre fixe la largeur maximale du rectangle bleu de la figure qui matérialise la distance entre deux boîtes englobantes des mots (voir figure 4.19). Les boîtes englobantes des mots sont les rectangles rouges qui entourent les lettres.

FIGURE 4.19 – Visualisation des boîtes englobantes des mots (rouge) et de l'espace entre les boîtes (bleu)

Le tampon local

La deuxième sous-tâche est une tâche système qui traite directement le résultat de l'étape précédente. Au début, le tampon est rempli avec le premier mot identifié lors de l'étape précédente, ensuite 2 cas peuvent se produire.

Soit la boîte englobante du mot courant est à une distance inférieure à celle fixée par l'utilisateur, dans ce cas, le mot est mis dans le tampon. Soit la distance est supérieure, alors le tampon est passé à l'étape fouille puis il est vidé et le mot suivant est traité.

4.2.4.2 La fouille

La fouille dans le tampon consiste à assembler les mots les uns avec les autres et créer ainsi une entité de plus haut niveau qui est la phrase. Cette entité de plus haut niveau conserve les liens vers tous les mots de l'étape précédente qui ont servi à créer cette phrase.

4.2.4.3 L'interprétation et l'évaluation

L'interprétation et l'évaluation sont très proche de celle des mots.

La figure 4.20, page 90, affiche le résultat de l'identification des phrases. L'écran se divise en 2 grandes parties, la partie de gauche où est affiché l'ensemble des phrases reconstituées et la partie droite qui visualise la phrase dans le folio quand l'utilisateur clique sur une phrase. Elle est entourée d'un cadre rouge qui visualise la boîte englobante de la phrase.

Avec l'interface de modification des phrases, l'utilisateur (voir figure 4.21, page 90) peut :

— fusionner 2 phrases, bouton « *Join the row with previous row* »,
— couper une phrase en deux, bouton « *Cut the string before the caracter* »,

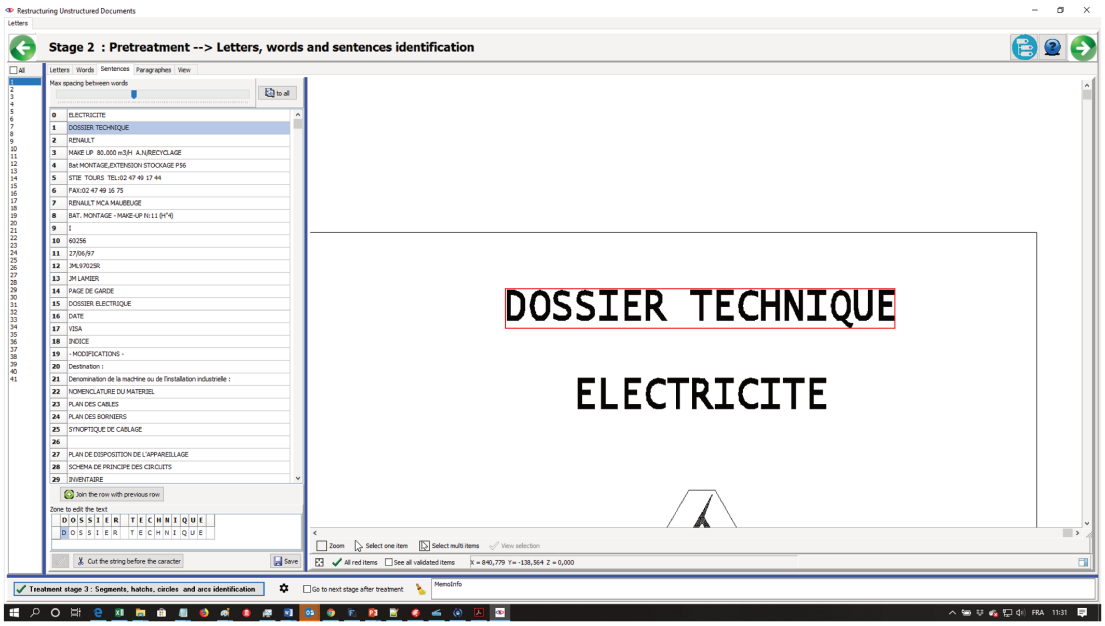


FIGURE 4.20 – Exemple de visualisation et de modification des phrases

— modifier les lettres de la phrase à l'aide de la zone « *Zone to edit the text* », si une ou plusieurs lettres sont modifiées et que l'utilisateur valide ce choix, ces modifications s'appliquent à toutes les lettres associées aux glyphes des lettres modifiées,
— sauver l'ensemble des opérations réalisées (fusion, coupure, modification) avec le bouton « *Save* ».

Comme on peut le constater, les tâches à réaliser ainsi que l'interface utilisateur pour l'interprétation et l'évaluation des informations sont exactement les mêmes que pour les mots.

4.2.5 Étape 2-4 : Restructuration des paragraphes

La restructuration des paragraphes est l'étape incrémentale qui suit immédiatement la restructuration des phrases. Cette étape incrémentale va lier les objets graphiques textuels (les phrases), les uns avec les autres pour former des

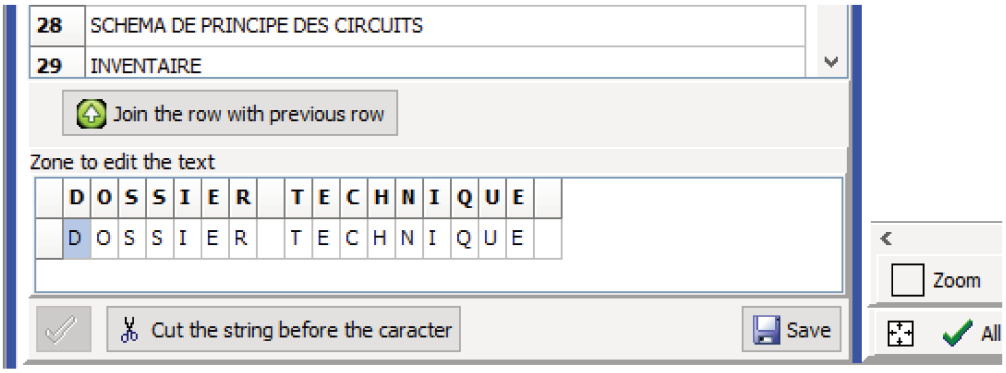


FIGURE 4.21 – Interface de modification des phrases

unités lexicales qui, dans le cas présent, sont les paragraphes(voir figure 4.14, page 79, quatrième et dernière colonne).

La restructuration des paragraphes est exactement la même que celle des phrases et des mots.

4.3 Étape 3 à 5 : Restructuration du document

La restructuration du document comprend 3 grandes étapes :

La restructuration des symboles et des équipotentielles (voir figure 4.22) qui s'effectue simultanément à partir de tous les éléments restructurés et des autres éléments graphiques du fichier d'entrée.

La restructuration des fonctions qui se fait immédiatement après la restructuration des symboles et des équipotentielles.

Puis la création du sommaire du dossier et la ventilation des fonctions dans le sommaire du dossier.

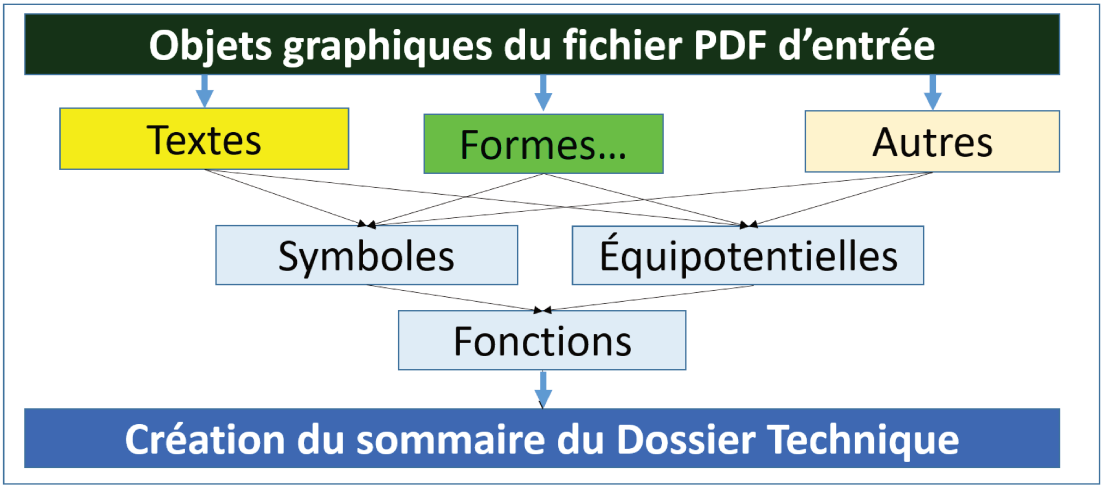


FIGURE 4.22 – Étapes de restructuration du dossier

4.3.1 Étape 3 : Restructuration des symboles et liaisons

La restructuration des symboles et des liaisons (ou équipotentielles) est une restructuration globale, c'est à dire qu'elle est réalisée sur l'ensemble du dossier. Les symboles ne peuvent être identifiés de façon précise que sur l'ensemble du dossier. Si l'on prend l'exemple du symbole de la protection (voir figure 1.2, page 8), ce symbole est présent 2 fois dans le folio, mais dans le dossier il peut être présent plus de 20 fois. Et comme nous l'avons vu dans le chapitre 3.6.2, page 59, plus le nombre de répétitions à identifier est important, plus l'algorithme « SAP » est rapide.

4.3.1.1 La codification

C'est la tâche de codification la plus complexe. Elle est la plus importante de l'ensemble des codifications mises en place pour restructurer les informations graphiques vectorielles.

Les paramètres

La première sous-tâche est une tâche utilisateur. L'utilisateur fixe deux paramètres :

— le paramètre du nombre de répétitions que l'algorithme « SAP »va identifier. Ce paramètre est attribué à T_{inf} lors de l'exécution de l'algorithme « SAP ». La valeur par défaut, prise par le système, est de 15, l'utilisateur peut changer cette valeur (elle sera proposée lors du prochain traitement),

— le paramètre du nombre d'éléments graphiques qui composent le symbole. Ce paramètre lié au paramètre précédent a une très forte influence sur les temps d'exécution de l'algorithme « SAP ». Plus ces paramètres sont grands, plus les temps d'exécution de l'algorithme « SAP » sont courts. Ce paramètre est attribué à L_{inf} lors de l'exécution de l'algorithme « SAP ». La valeur par défaut, prise par le système, est de 15. L'utilisateur peut changer cette valeur (elle sera proposée lors du prochain traitement).

L'utilisateur fixe ces deux paramètres avec les deux curseurs comme pour les autres restructurations vectorielles.

Le bouton, situé à la droite des deux curseurs, permet d'affecter les valeurs de ces deux curseurs à l'ensemble des folios du dossier.

Le tampon local

La deuxième sous-tâche est une tâche système qui codifie les objets graphiques du fichier d'entrée. Cette sous-tâche est au coeur du système de restructuration.

4.3.1.2 La fouille

La fouille s'effectue avec l'algorithme 2 que nous avons décrit page 59. Cet algorithme subit une légère modification car. Quand l'algorithme « SAP » rencontre le code de valeur 0, dans un des suffixes, il ne doit pas traiter ce code et il ne doit pas traiter les codes qui suivent le code de valeur 0 dans le suffixe. La récursivité de l'algorithme s'arrête.

Nous présentons l'algorithme non optimisé de l'algorithme « SAP » qui tient compte de cette spécificité et que nous avons nommé « SAPSymbole ».

Algorithm 5 SAPSymbole(R,p)

$bProfond := True$ {Indicateur de profondeur est mis à vrai}
For $s \in R$ **do** $B[S[p]] := B[S[p]U\{s\}$ {Range les suffixes dans les paquets}
For $c \in \Sigma$ **do** {Pour chaque code : si le paquet n'a pas assez de suffixes}
if $(|B[c]| \geq T_{inf})and(p \leq L_{inf})and(c \neq 0)$ **then** {et si profondeur $<$ limite }
  $SAPSymbole(B[c], p+1)$ {et si $C \neq 0$, alors appel récursif}
if $ConditionsPourEnregistrer$ **then** {Si les conditions sont vraies}
  $EnregistreRepetition$ {alors on enregistre la répétition}
$bProfond := False$ {Indicateur du paquet le plus profond est mis à faux}

Une fois la fouille terminée, les répétitions qui ont été identifiées sont com-

parées aux codifications des symboles qui ont déjà été enregistrées dans la base de connaissances. Cette comparaison nous donne 2 ensembles : l'ensemble des répétitions qui correspond à des symboles déjà identifiés, et l'ensemble des répétitions pour lesquelles nous supposons que la répétition correspond à un symbole. C'est l'utilisateur qui, lors de la phase d'interprétation et d'évaluation, va confirmer ou infirmer les choix proposés par le système.

4.3.1.3 L'interprétation et l'évaluation

L'interprétation et l'évaluation sont différentes suivant l'ensemble auquel appartient la répétition. Nous allons détailler les actions que doit faire l'utilisateur en fonction de l'ensemble des répétitions qu'il traite.

Répétitions non trouvées dans la base de connaissances

Pour les répétitions non trouvées dans la base de connaissances, trois cas peuvent se produire :

— La répétition ne correspond pas à un symbole : l'utilisateur supprime la répétition de l'ensemble.
— La répétition correspond à un symbole déjà présent dans la base, mais sa forme graphique est différente : il enregistre la répétition comme nouvelle instance du symbole dans la base de connaissances.
— La répétition correspond à un nouveau symbole : il l'enregistre dans la base de connaissances (Annexe sur la restructuration des symboles n'est pas diffusée, la société 1A3i, qui finance cette thèse, ne le souhaite pas).

Répétitions trouvées dans la base de connaissances

Pour les répétitions trouvées dans la base de connaissances, l'utilisateur va vérifier, pour chaque répétition, si le système a fait le bon choix pour toutes les occurrences de chaque répétition.

Pour chaque occurrence de chaque répétition, l'utilisateur a trois possibilités :

— Valider les choix du système : la répétition est considérée comme un symbole.
— Invalider les choix du système : l'ensemble des éléments graphiques de la répétition sera retraité à l'itération suivante.
— Modifier les choix faits par le système : cette modification consiste à réaffecter les textes aux paramètres du symbole.

Traitement des équipotentielles (liaisons)

Une fois que le l'utilisateur a validé les symboles d'une itération, tous les segments de droite qui arrivent sur les points de connexion des symboles, sont transformés en équipotentielles. Et comme pour les symboles, ils seront associés au code de valeur 0 lors de l'itération suivante.

Pour terminer, l'utilisateur peut relancer une nouvelle itération en modifiant les curseurs et donc les paramètres de traitement.

La stratégie est de commencer par les symboles qui sont les plus nombreux

et composés de plus d'éléments, puis de diminuer progressivement ces deux valeurs. Cette stratégie permet d'avoir des temps de réponse très courts de l'algorithme « SAPSymbole ». En effet, au début, il y a beaucoup d'éléments en entrée, mais les 2 seuils T_{inf} et $L - inf$ sont élevés. Ces 2 seuils permettent d'avoir un temps de réponse très court de l'algorithme « SAPSymbole ». Á la fin, quand ces limites diminuent, le nombre d'éléments en entrée a considérablement diminué, car les symboles et les équipotentielles sont identifiés et remplacés par le code 0, ce qui stoppe très rapidement la récursivité.

4.3.2 Étape 4 : Restructuration des fonctions

La restructuration des fonctions est globale. L'objectif est de trouver les séquences de symboles qui se répètent d'un folio à l'autre ou à l'intérieur d'un même folio.

L'utilisateur fixe avec un curseur le nombre minimum de symboles que doit contenir une fonction.

Pour identifier les fonctions, nous prenons tous les symboles issus de l'étape de restructuration précédente et nous créons un arbre où les noeuds sont les symboles et les équipotentielles sont les arcs. Cet arbre est créé en respectant le positionnement du symbole sur le folio. Puis nous parcourons tous les chemins de l'arbre pour arriver aux feuilles.

Et nous regardons avec l'algorithme SAP s'il existe des répétitions.

4.3.2.1 La codification

Les chemins sont formés des noeuds de l'arbre qui correspondent aux symboles. Chaque symbole est associé à un numéro N qui correspond à l'ordre chronologique de création de ces symboles dans la base de connaissances. Nous obtenons le code du noeud dans le chemin en appliquant la formule :

$code = (N \mod 255) + 1$

Après cette codification, les chemins de l'arbre sont rangés dans le tampon et séparés par le code de valeur 0.

4.3.2.2 La fouille

Nous utilisons l'algorithme « SAPSymbole » pour identifier les répétitions de la même façon que pour les autres étapes incrémentales.

4.3.2.3 L'interprétation et l'évaluation

L'interprétation et l'évaluation sont les mêmes que pour les symboles, à la seule différence que l'on manipule des fonctions au lieu de symboles, le principe est exactement le même.

4.3.3 Étape 5 : Restructuration du dossier

La restructuration du dossier est la dernière étape de restructuration. Elle consiste à identifier dans les cartouches des folios, les sommaires et les sous-sommaires, ce qui est implicite une fois que l'on a identifié le symbole du cartouche dans le folio. Cette opération nous donne une arborescence sommaire, sous-sommaire.

Ensuite le système range chaque fonction identifiée à l'étape précédente dans l'arborescence sommaire sous-sommaire.

L'utilisateur peut modifier en totalité cette organisation proposée par le système.

Section 5

Résultats

Les tests sont faits sur un PC Dell Précision M6800 avec le processeur Intel™ Core™ i7-4800 MQ CPU 2.70 Ghz, mémoire physique RAM 16 Go, mémoire virtuelle 32 Go, dont le système d'exploitation est Windows™ 10.0.17134.

De la méthode KDD à (A)KDD

Le passage de la méthode KDD à (A)KDD est sûrement le point le plus important de tous ces travaux. Le développement d'une méthode et d'un modèle pour traiter l'ensemble des données en entrée de la restructuration des documents graphiques sont les deux points essentiels pour traiter des données brutes. Nous l'avons dit, à notre connaissance, aucune thèse n'a traité ce problème de la restructuration des documents graphiques contenus dans des fichiers vectoriels comme les fichiers PDF. Pour aborder ce nouveau champs d'application, il nous a paru indispensable de définir une méthode et un modèle.

Comme l'indiquaient les auteurs dans l'article « *Visual Analytics : Definition, Process and Challenges* »[Kei+08], « L'acquisition de données brutes n'est plus le problème : c'est la capacité à identifier des méthodes et des modèles qui peuvent transformer les données en connaissances fiables et prouvables ». C'est ce que nous avons fait avec la méthode (A)KDD : nous avons identifié une méthode, et nous l'avons adaptée à notre problème. Pour le modèle, nous le verrons au paragraphe suivant.

Nous avons donc repris les travaux de 1996, d'Usama Fayyad, Gregory Piatetsky-Shapiro et Padhraic Smyth qui avaient pour but de développer une méthode pour extraire les connaissances des bases de données : la méthode KDD [FPS96b]. C'est cette méthode KDD que nous avons reprise et adaptée pour en faire notre méthode (A)KDD, qui, grâce à l'implication de l'utilisateur tout au long du processus et à l'aspect incrémental, permet de restructurer les documents graphiques.

Dans leur article « *The KDD process for extracting useful knowledge from volumes of data* », ils concluent par « La réalisation d'applications réelles agira comme un filtre pour passer au crible les bonnes théories et techniques de celles moins utiles ». C'est pour cette raison que, pendant ces travaux de thèse,

nous avons développé un prototype qui reprend l'ensemble des théories et des techniques que nous avons développées pour les valider sur un cas réel.

Cette adaptation de la méthode est la première contribution fondamentale sur laquelle tous les travaux ont été développés.

Le modèle des données

Passage du 2D au 1D

Le modèle, c'est la deuxième contribution fondamentale de ces travaux. Après ou, plus exactement, avec la méthode, il fallait définir un modèle de représentation des données. En règle générale, quand on traite des données graphiques on utilise un modèle « 2D » et parfois « 3D » pour manipuler les informations. Nous, nous avons fait la pari de transformer ces informations « 2D » en un modèle « 1D ». Nous avons transformé et codé tous les objets graphiques en une chaine de code, avec plusieurs types de codifications pour être insensibles à la translation et, si on le souhaite, à la rotation et au changement d'échelle.

Déjà en 1974 Freeman [Fre61], dans le but de compresser les images, avait proposé de remplacer le contour d'une forme, dans une image, par une chaîne de code (code 4 ou 8 caractères, voir figure 5.1) avec la possibilité de le compresser avec les techniques d'HUFFMAN [Huf52] .

3	2	1
4	X	0
5	6	7

	1	
2	X	0
	3	

FIGURE 5.1 – Code de Freeman à 4 ou 8 directions. X est le point courant, les chiffres correspondent aux 4 ou 8 directions du point voisin appartenant à la frontière de l'objet.

Papert, en 1973, propose de piloter sa tortue [Pap73] pour suivre un contour avec 2 codes (0 et 1). Le 0 pour aller à droite, et, le 1 pour aller à gauche, sinon aller tout droit. Le code de Papert est invariant par rotation et invariant par translation, il suffit de déplacer et d'orienter la tortue.

Nous, nous avons fait exactement la même chose, pas uniquement pour coder le contour d'une forme, mais pour tous les objets graphiques. Ce sont ces techniques de codifications que nous avons appliquées à l'ensemble des fichiers d'entrée pour passer d'un modèle « 2D » à un modèle « 1D ».

Ordre chronologique

Le passage du modèle « 2D » au modèle « 1D », n'est possible que si les éléments sont bien rangés dans un ordre chronologique dans le fichier d'entrée. Et surtout il faut que cet ordre chronologique, existant dans le fichier d'entrée, ne soit pas détruit par des outils de relecture. C'est pour cette raison que nous

avons développé nos propres outils de lecture des différents fichiers vectoriels comme les fichiers PDF. Et que ces outils ont été développés en amont des travaux de cette thèse.

C'est grâce à cet ordre chronologique que nous avons pu développer tous nos algorithmes de codification et de fouille de données. Et, encore une fois, c'est la première fois (à notre connaissance) que l'ordre chronologique a été pris en compte pour restructurer des documents PDF contenant des graphiques. Dans tous les travaux que nous avons pu consulter, pour restructurer des fichiers PDF qui contiennent des textes, des tableaux, des formulaires, ou des formules mathématiques, aucun n'utilise l'ordre chronologique.

Evaluation de l'algorithme SAP

Nous avons une méthode, nous avons un modèle, maintenant il faut vérifier que les algorithmes qui exploitent ces données sont performants. La performance que l'on recherche, n'est pas la performance pour la performance. Notre objectif de performance est de rendre les temps d'attente entre 2 actions inférieurs à ce que peut accepter un utilisateur. Prenons un exemple, si le temps de traitement prend 10 secondes et que l'optimisation le divise par 20, nous aurons un temps d'attente d'une demie seconde, ce qui est acceptable par l'utilisateur.

Nous avons testé l'extraction des répétitions en utilisant les mêmes fichiers que ceux de Puglisi pour avoir un référentiel de comparaison. Le premier test que nous avons fait concerne le temps de traitements du tri par paquets de l'algorithme 1. Nous l'avons testé sur le fichier « *HowTo* ». Le fichier « *HowTo* » est accessible à l'adresse http ://www.cas.mcmaster.ca/~bill/strings/.

Des travaux de Puglisi [PSY10], nous avons extrait les résultats (voir figure 5.2) qu'il obtient pour le traitement du fichier « *HowTo* » (voir les résultats complets page 29). Les algorithmes, PSY_{1-1} à PSY_{1-4}, développés par Puglisi pour extraire les répétitions, nécessitent la création du tableau des suffixes (SA), des plus longs préfixes communs (LCP) et de 2 autres tableaux (BWT et LAST). On constate, cependant, dans les résultats obtenus (Figure 5.2), que la somme des temps pour créer les 4 tableaux préparatoires (1,912+0,17,+0,035+0,039) est égal à 144 fois le temps de l'algorithme (0,015) le plus rapide PSY1-1 pour détecter les répétitions.

Temps des traitements (en microsecondes)								
Fichier	Prétraitement des tableaux					Les algorithmes PSY1-1 à PSY1-4		
	SA	LCP	BWT	LAST		PSY1−1	PSY1−2	PSY1−3	PSY1−4
howto(*)	1,912	0,178	0,035	0,039		**0,015**	0,017	0,018	0,016
(*) le fichier "Howto" appartient à la collection des fichiers accessibles à l'adresse http://www.cas.mcmaster.ca/~bill/strings/								

FIGURE 5.2 – Extrait des résultats de PUGLISI et al. [PSY10] (temps en microsecondes)

Dans l'algorithme de tri par paquets nous avons traité deux cas :

— arrêt du tri par paquets pour être remplacé par le tri par insertion à partir d'un seuil qui représente le nombre de suffixes dans un paquet,
— arrêt complet du tri des paquets à partir d'un seuil qui représente le nombre de suffixes dans un paquet.

Tri par paquets du fichier HowTo		
	Temps en microecondes par caractére	
Seuil	Tri Insertion	Pas de Tri
256	1,308	0,111
128	1,155	0,123
64	1,112	0,154
32	1,121	0,204
16	1,189	0,285
8	1,298	0,442

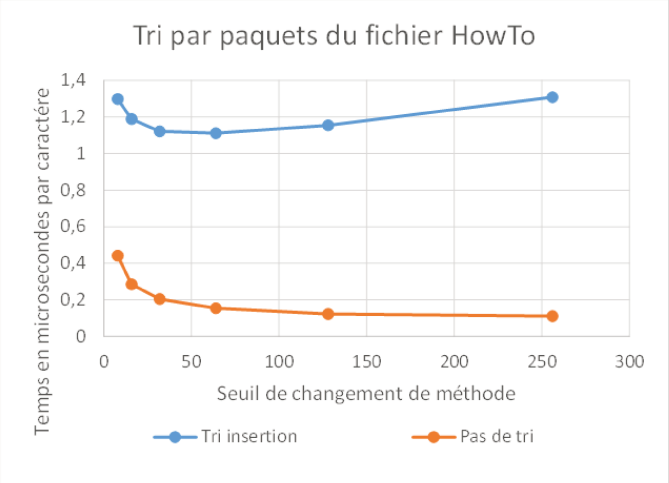


FIGURE 5.3 – Temps des tris par paquets (en microsecondes) du fichier « *HowTo* » en fonction d'un seuil de changement de méthode

Dans la Figure 5.3, la colonne « Tri Insertion », correspond au premier cas mentionné ci-dessus. Cette colonne est conforme aux résultats de SEDGEWICK et al. [SW15], et montre la nécessité de changer de tri quand la taille du paquet est petite. La première colonne contient les valeurs des seuils (c'est à dire le nombre de suffixes d'un paquet) qui déclenche le changement de méthode de tri. La colonne « Pas de Tri » correspond au deuxième cas mentionné ci-dessus. Là encore, on vérifie très bien les propos de SEDGEWICK et al. [SW15]. Ce qui prend du temps, c'est le tri des paquets avec très peu d'éléments. Ceci confirme parfaitement l'option que nous avons prise de ne pas trier tous les suffixes car, pour identifier les répétitions, il n'est pas nécessaire de trier les suffixes de la répétition.

Dans notre algorithme SAP, le temps pour identifier les répétitions qui ont un nombre de suffixes supérieur à 16 (T_{inf}) et une longueur de sous-chaîne supérieure à 8 (L_{inf}) est de 0,115 microsecondes (voir figure 5.4, page 101). Ce temps est à comparer aux 1,189 microsecondes du tri complet par paquets de tous les suffixes (voir figure 5.3, 100) : le temps de traitement est divisé par 10. Comparé à ceux de PUGLISI et al. [PSY10] il est 19 fois plus rapide (voir figure 5.2, page 99). L'espace mémoire occupé par notre algorithme 2 est légèrement supérieur aux algorithmes qui utilisent les tableaux des suffixes, car, dans notre cas, il faut rajouter l'espace occupé par les compteurs : soit 1024 octets pour les compteurs ainsi que 4 fois la taille du plus grand paquet, plus le fichier en entrée.

Avec l'algorithme 3, on diminue l'espace total occupé à 8 fois la taille du plus grand paquet. La taille du plus grand paquet est presque égale à $N/32$. Donc l'espace mémoire utilisé est égal à $N/4$, plus l'espace des compteurs, plus

l'espace du fichier d'entrée N, soit un total de $1.25N$. Pour rappel, l'espace mémoire utilisé par le tableau des suffixes est égal à $5N$. Notre solution prend 4 fois moins d'espace mémoire que le calcul du tableau des suffixes. Mais l'algorithme 3 est moins rapide de 10% que l'algorithme 2.

Algorithme "SAP" sur le Fichier HowTo							
		L_inf : Longueur des répétitions					
		8	16	32	64	128	256
T_inf : Nombre des suffixes dans un paquet	256	0,084	0,093	0,095	0,097	0,102	0,103
	128	0,087	0,101	0,108	0,115	0,120	0,125
	64	0,093	0,123	0,136	0,140	0,145	0,146
	32	0,108	0,140	0,161	0,170	0,183	0,194
	16	0,115	0,180	0,208	0,227	0,242	0,258
	8	0,154	0,245	0,296	0,332	0,356	0,388
		Temps en microcondes par caractére					

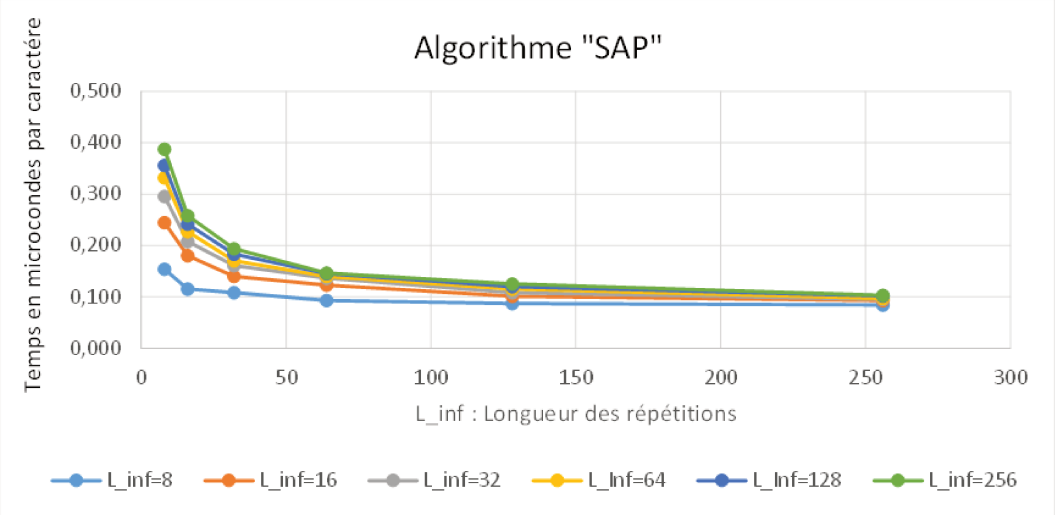


FIGURE 5.4 – Temps d'exécution en microsecondes de l'algorithme « SAP » sur le fichier « *HowTo* »

Ces résultats confirment l'intérêt de développer une nouvelle méthode de détection des motifs à l'intérieur d'une chaîne qui fusionne en un seul algorithme plusieurs opérations, qui, dans de nombreux travaux de recherches (comme ceux de Puglisi), se font séquentiellement. La création du tableau des suffixes, puis la création du tableau des LCP, puis la création des 2 tableaux (BWT et LAST) et pour terminer les algorithmes de détection des motifs PSY_{1-1} à PSY_{1-4} font que le cumul de toutes ces opérations prend 20 fois plus de temps que notre algorithme SAP.

L'utilisation d'un tel algorithme n'est possible que grâce à la présence de l'utilisateur et un tel algorithme n'est nécessaire que parce que l'utilisateur est présent. L'un ne va pas sans l'autre. Le fait que l'utilisateur interagisse avec l'algorithme SAP nécessite des temps de réponse très courts compatibles avec le temps d'attente acceptable par un utilisateur.

La méthode (A)KDD est centrée utilisateur, c'est lui qui fixe les seuils T_{inf} et L_{inf} de l'algorithme SAP, grâce à deux paramètres qui déterminent le nombre de répétitions recherchées (T_{inf}) et la longueur des répétitions (L_{inf}), Ce sont ces deux paramètres qui permettent de diminuer les temps de recherche de façon très significative.

Grâce à l'algorithme SAP, la stratégie est très simple. Elle consiste à identifier en premier les symboles présents le plus grand nombre de fois et/ou les plus importants en nombre de motifs, puis, ceux qui apparaissent le moins souvent et/ou les plus petits. A chaque itération, les symboles identifiés sont supprimés du fichier d'entrée. De cette façon l'itération suivante se fait sur un fichier de taille inférieure, elle est plus rapide car nous avons moins d'objets en entrée.

Nous présentons maintenant les résultats de l'algorithme SAP optimisé qui optimise l'espace mémoire mais qui en temps est environ 10% mois rapide que la version de l'algorithme SAP non optimisé

Algorithme "SAP" optimisé sur le Fichier HowTo							
		L_inf : Longueur des répétitions					
		8	16	32	64	128	256
T_inf : Nombre des suffixes dans un paquet	256	0,092	0,102	0,105	0,107	0,112	0,113
	128	0,096	0,111	0,119	0,127	0,132	0,138
	64	0,102	0,135	0,150	0,154	0,160	0,161
	32	0,119	0,154	0,177	0,187	0,201	0,213
	16	0,127	0,198	0,229	0,250	0,266	0,284
	8	0,169	0,270	0,326	0,365	0,392	0,427
		Temps en microcondes par caractére					

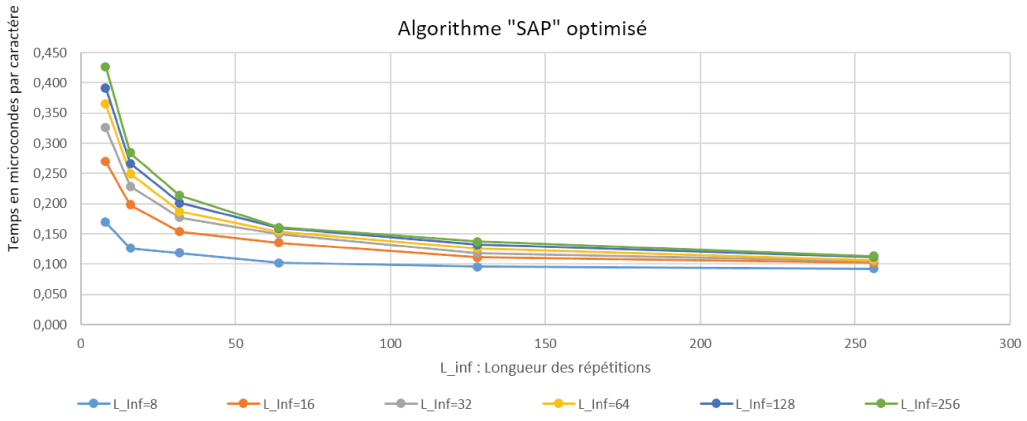


FIGURE 5.5 – Temps d'exécution en microsecondes de l'algorithme « SAP » optimisé sur le fichier « *HowTo* »

Résultats pour la méthode (A)KDD

Cette préoccupation de diminution du nombre des objets en entrée des algorithmes est la préoccupation permanente de l'ensemble de ces travaux. C'est

cette préoccupation qui nous a conduits à mettre en place les étapes incrémentales de la restructuration. On progresse pas à pas, et, à chaque progression, le nombre d'éléments à traiter diminue pour tendre vers zéro.

Nous avons relevé les valeurs du nombre d'éléments en entrée de deux folios, et le nombre d'éléments aux grandes étapes de la restructuration à la fin de la restructuration vectorielle et à la fin de la restructuration textuelle, et, pour terminer, le nombre de symboles et d'équipotentielles identifiés (voir figure 5.6, page 103). Nous aurions pu faire ce tableau pour l'ensemble des folios, mais nous avons préféré donner seulement 2 exemples pour nous concentrer sur les ratios.

Les chiffres que nous donnons concernent les deux folios que nous avons déjà utilisés lors de tests de relecture du fichier PDF par Microsoft™ et Adobe™. Il s'agit du folio de la page de garde en page 31 et du folio des moteurs en page 32.

		Folio 1 Page de garde	Folio 4 Folio des moteurs
Nombre d'éléments en entrée dans le fichier PDF	**E1**	**20 830**	**7888**
Nombre d'éléments supprimés lors du nettoyage		16 143	6232
Reste après nettoyage	**R1**	**4687**	**1656**
Nombre d'éléments supprimés lors de la restructuration des objets vectoriels		3231	984
Reste après restructuration vectorielle (Étape 1)	**V1**	**1456**	**672**
Nombre d'éléments supprimés lors de la restructuration des objets textuels		1378	446
Reste après restructuration textuelle (Étape 2)	**T1**	**78**	**226**
Nombre d'éléments supprimés lors de la restructuration des symboles/liaisons		76	188
Reste après restructuration des symboles/liaison (Étape 3)	**S1**	**2**	**38**
Ratio E1/T1		267,1	34,9
Ratio R1/T1		60,1	7,3

FIGURE 5.6 – Nombre d'éléments après les principales étapes de restructuration

L'analyse de ces chiffres montre l'importance et surtout la nécessité de nettoyer le fichier d'entrée. Le ratio E1/T1 (nombre d'éléments en entrée sur nombre d'éléments après restructuration vectorielle et textuelle) est, pour la page de garde, proche de 267 et, pour le folio des moteurs, proche de 35.

Le ratio R1/T1 (nombre d'éléments après nettoyage sur nombre d'éléments après restructuration vectorielle et textuelle) est aussi très intéressant : dans le premier cas il est de 60, et dans le deuxième de 7.

Nous avons, déjà, souligné l'importance du nettoyage des données et nous avons dit qu'une de nos contributions était dans ce nettoyage. Après nettoyage on a divisé par 5 (presque) le nombre des éléments en entrée.

Après la restructuration des éléments vectoriels de l'étape 1 et celle des éléments textuels de l'étape 2, nous avons divisé le nombre d'éléments par des coefficients compris entre 60 et 7, ce qui fait que les temps de réponses de l'ensemble est acceptable par l'utilisateur.

Autre point important : sur le folio de la page de garde on part de plus de

20 000 éléments pour arriver à 2 symboles : le logo Renault et le cartouche.

Prenons par exemple le cas où une étape permet de diviser par 10 le nombre d'objets entre le nombre d'objets en entrée et le nombre d'objets en sortie. Si l'algorithme de l'étape suivante est linéaire, le fait de traiter 10 objets au lieu de 100 divise le temps de traitement de l'algorithme par 10. Si l'algorithme est en $Nlog_2(N)$ le temps est divisé par 33.21, et, si l'algorithme est en N^2, le temps est divisé par 100. Dans ce dernier cas, on divise par 100 le temps d'attente de l'utilisateur, ce qui est loin d'être négligeable.

Section 6

Conclusions et perspectives

Je citerai la conclusion d'Usama Fayyad en 1996 dans l'article « *The KDD Process for Extracting Useful Knowledge from Volumes of Data* » [FPS96b], article ou il définit la méthode KDD : « *KDD has evolved, and continues to evolve, from the intersection of research fields such as machine learning, pattern recognition, databases, statistics, AI, knowledge acquisition for expert systems, data visualization, and high-performance computing. The unifying goal is extracting high-level knowledge from low-level data in the context of large data sets* ». Le travail que nous avons fait s'inscrit dans l'extraction des connaissances graphiques, et, comme le souligne Usama Fayyad, il nécessite la maîtrise de nombreuses disciplines auxquelles nous pouvons rajouter la stringologie, la DAO et la CAO.

La méthode (A)KDD

Le but ultime est d'apporter la puissance de la méthode (A)KDD dans chaque ordinateur des bureaux techniques de la « Smart factory ». Ceci pour permettre une meilleure exploration des données, plus rapide et plus intuitive. L'objectif final étant d'améliorer les performances économiques de ces entreprises.

Dans ce travail, nous avons décrit le passage de la méthode KDD à la méthode (A)KDD, en appliquant les 4 phases fondamentales de la méthode à la restructuration des données graphiques, et, surtout nous avons détaillé le côté incrémental de la méthode (A)KDD. Nous avons vu l'importance des ontologies pour obtenir un résultat efficace et précis, nous avons bien précisé le rôle fondamental de l'utilisateur quand le volume des données en entrée est faible.

Mais, surtout, nous avons utilisé une méthode qui repose sur le fait que les éléments graphiques dans un fichier PDF sont dessinés toujours de la même façon. Cette hypothèse et sa vérification sur le terrain, nous ont permis de développer des algorithmes qui reposent sur le fait que les éléments suivent un ordre chronologique bien précis. Les temps de réponse de ces algorithmes sont très courts, car souvent, il suffit de regarder l'élément précédent ou l'élément suivant pour prendre une décision. Cette rapidité fait que nous avons pu mettre l'utilisateur au coeur de la méthode, les temps d'attente deviennent acceptables

pour l'utilisateur.

L'algorithme SAP

Nous avons décrit un algorithme très simple, très rapide et très peu gourmand en place mémoire qui supprime l'utilisation des tableaux SA et LCP pour trouver les répétitions dans une chaîne. Nous avons également profité au maximum des possibilités offertes par les mémoires caches pour diminuer les temps d'exécution pour développer algorithme optimisé.

La dissémination

L'ensemble de ces travaux a été présenté :
— à la conférence SMART- 2017 à Venise (article)[PC17],
— à la conférence KDD 2017 à Hallifax (aux experts qui recevaient les doctorants). Lors de cette conférence j'ai présenté mes travaux au Professeur Usama Fayyad, je lui ai expliqué mon souhait d'appliquer la méthode KDD, qu'il avait développé en 1996, à la fouille de données graphiques pour les restructurer. L'idée lui a paru originale, mais il était étonné que j'utilise cette méthode qui lui paraissait ancienne.
— à la conférence ICAI'18. Las Vegas (article)[Pér19b],
— à la conférence KDD 2018 à Londres (aux experts qui recevaient les doctorants),
— à la conférence ECML-PKDD de 2018 à Dublin (à l'ensemble des partipants),
— à la conférence EGC-2019 à Metz (article)[Pér19b],
— au CES 2018 (Consumer Electronics Show) à Las-Vegas (prototype)[Pér18c],
— au CES 2019 (Consumer Electronics Show) à Las-Vegas (prototype)[Pér19a],
— au SOFINS 2019 (Spécial Opérations Force Innovation Network Séminar) au camp de Souge Bordeaux (prototype),
— à NAIA 2019 (forum Néo-Aquitain de l'Intelligence Artificielle) à Bordeaux (prototype).

Le prototype

A titre indicatif, l'ensemble du prototype sur lequel s'appuient ces travaux a été développé en Pascal Objet avec l'outil de développement Delphi™ de la société Embarcadero™. Aujourd'hui, l'ensemble du développement représente plus de 300 000 lignes de code assemblées et testées qui se répartissent de la façon suivante :
— 100 000 lignes développées pendant la phase de préparation de la thèse (2 ans) dont l'objectif était de lire différents types de formats de fichiers vectoriels (voir annexe D.3) et principalement les fichiers PDF.
— 100 000 lignes développées pendant la thèse pour valider les hypothèses et développer tous les algorithmes et toutes les interfaces qui ont permis

de vérifier l'efficacité de la méthode (A)KDD.
— 100 000 lignes de librairie et de composants (achetés avec les sources) qu'il a fallu tester et parfois adapter à nos besoins.

A raison de 60 lignes par page, l'ensemble du code développé et testé représente environ 33 fois le volume de cette thèse. L'ensemble de 300 000 lignes de codes a été développé et mis au point sur une période de 5 ans.

Ces travaux de recherche ont fait l'objet d'un dépôt de brevet en France en Juillet 2018, et à l'international en 2019, « PROCÉDÉ DE RESTRUCTURATION D'OBJETS GRAPHIQUES » [Pér18a].

Les évolutions de l'algorithme SAP

L'agorithme SAP peut être amélioré sur deux points principaux.

Le premier point concerne les compteurs : dans le tri par paquets nous nous sommes limités à 256 codes, mais on peut imaginer des codifications plus fines qui nécessitent des codifications sur plus de 8 bits, par exemple 12 bits ou 16 bits. Avec de telles codifications, le nombre de compteurs devient très important. Pour un code à 12 bits, il faut 2^{12} compteurs, soit 4096 compteurs. Pour un code sur 16 bits, il faut au 2^{16}, soit 65 536 compteurs. Un tel nombre de compteurs ne peut pas être géré de la même façon que dans l'algorithme 2 et dans l'algorithme 3. Par contre, on constate que, dès que la récursivité de l'algorithme (la profondeur) dépasse 3 ou 4 itérations, le nombre de compteurs renseignés est très faible par rapport au nombre total des compteurs renseignés aux premières itérations. Nous avons commencé l'écriture de cet algorithme, mais il n'a pas été introduit dans le prototype utilisé dans le cadre de cette thèse.

Le deuxième point concerne le parallélisme : en parallélisant plusieurs tâches de l'algorithme SAP, nous pouvons diminuer les temps de traitement. Ce sont des optimisations que nous avons déjà réalisées avec l'Inria de Bordeaux pour un simulateur électrique [Ala+15]. Le parallélisme est très facile à mettre en oeuvre, il suffit de poursuivre le tri pour chaque paquet sur un *thread* différent, dans la limite des *threads* qui peuvent être exécutés sur la machine.

Les évolutions de la méthode (A)KDD

La méthode acrshort(A)KDD peut être améliorée suivant 2 axes.

Le premier axe : les interfaces utilisateurs. En fonction du retour des utilisateurs Il s'agit de les améliorer, et d'avoir la possibilité de visualiser très rapidement et de façon très claire et très précise un nombre encore plus grand de données graphiques. Dans le cadre de la thèse nous avons traité un dossier. Dans le cas d'un traitement simultané sur plusieurs dossiers il est nécessaire de pouvoir visualiser beaucoup plus d'informations Ceci nécessitera des développements nouveaux des principales interfaces graphiques de visualisation des données. Il faut conserver des temps d'affichage très courts, même avec beaucoup de données.

Le deuxième axe : les ontologies. Pour l'instant, dans le prototype, nous avons créé une ontologie pour traiter des dossiers de schématique électrique. Cette ontologie a été créée avec des outils qui ne sont pas conviviaux, mais qui peuvent être améliorés. Nous aurions pu prendre directement un produit comme le logiciel « Protégé » de l'Université de Stanford. Nous ne l'avons pas fait car nous étions dans le cadre d'un développement industriel, et rappelons le, c'est l'objectif de ces travaux. Il est difficile et même dangereux de développer des solutions dans lesquelles nous n'aurions pas une maîtrise complète du produit.

Les perspectives industrielles

L'objectif maintenant est d'industrialiser l'ensemble de nos résultats pendant une période de 2 ans, et, dans la mesure du possible, de poursuivre les travaux de recherche sur la méthode (A)KDD que nous avons développée pour la rendre encore plus rapide, plus conviviale et plus efficace.

Traiter un ensemble de dossiers d'une même source

Nous avons traité, dans le cadre de cette thèse, la restructuration d'un dossier, mais on peut très bien imaginer le traitement de tout un ensemble de fichiers provenant d'une même source, et, ne plus restructurer dossier par dossier, mais source par source. C'est pour cette raison que les points d'améliorations que nous avons mentionnés ci-dessus sont importants. On peut imaginer un ensemble de fichiers issus d'un logiciel comme See Expert™, par exemple de 50 à 100 dossiers. Tous ces dossiers sont issus du même logiciel, ce qui veut dire que le symbole de la bobine de relais est dessiné exactement de la même façon dans tous les dossiers, et, dans ce cas, ce n'est plus 40 mais 4000 fois que le symbole de la bobine sera présent dans l'ensemble des dossiers.

On imagine facilement, que multiplier par 100 la taille de la chaîne d'entrée, va nécessiter d'améliorer les algorithmes mis en place comme nous l'avons vu, et, entraîner des problèmes de visualisations des informations.

Traiter des domaines différents

Le sujet traité dans cette thèse est la schématique électrique en particulier, et la schématique de façon générale. Pour traiter ces différents domaines de la schématique, il faudra préparer les ontologies qui y correspondent. Bien sûr nous utiliserons les ontologies que nous avons développées pour la schématique électrique, mais il faudra les adapter et les compléter pour les autres domaines.

À côté de la schématique, il y a d'autres domaines très importants à traiter, comme les plans de bâtiments et les plans de mécanique en 2D et en 3D. Pour les plans de bâtiments, il faudra être plus vigilant aux orientations des symboles et, dans ce cas, prendre des codifications insensibles aux rotations, mais aussi aux changements d'échelle (par exemple pour les portes ou les fenêtres). Pour la mécanique, il faudra faire des développements complémentaires qui porteront sur les axes, et en particulier les axes de symétrie. il faudra aussi compléter les

développements sur les hachures car elles sont beaucoup plus nombreuses qu'en schématique et elles ont des significations plus importantes, en particulier pour les schémas en coupe.

On peut citer aussi le PID qui a une schématique un peu spéciale, car des éléments comme les cuves peuvent subir des changements d'échelle verticaux et/ou horizontaux qui ne sont pas nécessairement homothétiques.

S'interfacer avec des logiciels de DAO/CAO

La base graphique utilisée est au format DXF/DWG. Comme nous l'avons dit, c'est un format lu par de nombreux logiciels. Mais, nous savons pertinemment que chaque fois que l'on passe par un format intermédiaire on perd de l'information. Donc, si l'on veut être plus performant, il faut pouvoir proposer de créer directement les informations graphiques dans un format natif, compréhensible par le logiciel. Ceci est de la pure industrialisation avec beaucoup de marketing. La mise en oeuvre d'une telle solution est quand même liée à quelques problèmes de généricité non triviaux à régler.

Comparer des versions de fichiers

C'est une des applications qui peut être développée et que nous avons d'ailleurs déjà commencé à développer dans le cadre d'une preuve de concept avec la société COMAU. La comparaison, qui se fait élément graphique par élément graphique, peut être améliorée en appliquant ponctuellement la restructuration développée dans le cadre de cette thèse. C'est un travail d'industrialisation à réaliser.

Indexer les documents graphiques

Vu l'augmentation du volume des données, il devient quasi impossible de savoir quelles sont les informations contenues dans un dossier graphique. La restructuration que nous avons développée peut facilement permettre l'indexation de toutes les informations textuelles présentes dans un dossier. Ce point est important car il est quasiment impossible à un service technique de savoir dans quel dossier, telle ou telle référence de matériel est utilisée.

L'indexation des éléments graphiques purs, comme l'exemple que nous avons cité de la bobine du relais, est un peu plus complexe que l'indexation d'un texte. On peut très bien imaginer une indexation très rapide de ce type d'information sous le contrôle d'un utilisateur.

Il est possible de développer ces deux types d'indexation à partir des travaux présentés ici. Il faudra modifier certaines étapes, mais c'est tout à fait réalisable.

Faire des recherches textuelles et graphiques

C'est le complément des développements du paragraphe précédent. Une fois que l'on a indexé le texte et le graphisme, il faut développer les outils qui

permettront de faire des recherches sur le texte, des recherches sur des éléments graphiques, des recherches qui mêlent le texte et le graphisme.

Bibliographie

[Ala+15] M. Alaya, M. Faverge, X. Lacoste, A. Péré-Laperne, J. Péré-Laperne, P. Ramet et al. « Simul'Elec and PASTIX interface specifications ». In : (2015).

[AN94] A. Andersson et S. Nilsson. « A New Efficient Radix Sort ». In : *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. SFCS '94. Washington, DC, USA : IEEE Computer Society, 1994, p. 714-721. doi : 10.1109/SFCS.1994.365721.

[AM91] E. Angel et D. Morrison. « Speeding Up Bresenham ». In : *IEEE Computer Graphics and Applications* 6 (1991), p. 16-17.

[Ank02] M. Ankerst. « Report on the SIGKDD-2002 panel the perfect data mining tool : interactive or automated ? » In : *ACM SIGKDD Explorations Newsletter* 4.2 (2002), p. 110-111.

[BK08] H. Baars et H.-G. Kemper. « Management support with structured and unstructured data—an integrated business intelligence framework ». In : *Information Systems Management* 25.2 (2008), p. 132-148.

[BBH03] S. R. Bagley, D. F. Brailsford et M. R. Hardy. « Creating reusable well-structured PDF as a sequence of component object graphic (COG) elements ». In : *Proceedings of the 2003 ACM symposium on Document engineering*. ACM. 2003, p. 58-67.

[Bak12] J. B. Baker. « A linear grammar approach for the analysis of mathematical documents ». English. d_ph. University of Birmingham, juil. 2012.

[Bar+15] D. W. Barowy, S. Gulwani, T. Hart et B. Zorn. « FlashRelate : extracting relational data from semi-structured spreadsheets using examples ». In : *ACM SIGPLAN Notices*. T. 50. 6. ACM. 2015, p. 218-228.

[BI] J.-L. Bloechle et R. Ingold. « Restructuration physique et logique de documents électroniques textuels ». In : ().

[BZ14] D. Blostein et R. Zanibbi. « Processing mathematical notation ». In : *Handbook of Document Image Processing and Recognition* (2014), p. 679-702.

[Bre65] J. E. Bresenham. « Algorithm for computer control of a digital plotter ». In : *IBM Systems journal* 4.1 (1965), p. 25-30.

[Buc97] M. K. Buckland. « What Is a “Document" ? » In : *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE* 48.9 (1997), p. 804-809.

[Car07] J. Cardoso. « The Semantic Web Vision : Where Are We ? » In : *IEEE Intelligent Systems* 22 (2007), p. 84-88. doi : doi.ieeecomputersociety.org/10.1109/MIS.2007.97.

[Cau+19] M. Causse, F. Lancelot, J. Maillant, J. Behrend, M. Cousy et N. Schneider. « Encoding decisions and expertise in the operator’s eyes : Using eye-tracking as input for system adaptation ». In : *International Journal of Human-Computer Studies* 125 (2019), p. 55-65.

[COV17] A. Chafik, L. Oyhenart et V. Vigneras. « Simulation des réseaux de torons englobant des circuits électriques complexes ». In : 2017.

[CYH04] H. Cheng, X. Yan et J. Han. « IncSpan : incremental mining of sequential patterns in large database ». In : *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2004, p. 527-532.

[Che+07] V. Chevrin, O. Couturier, E. Mephu Nguifo et J. Rouillard. « Recherche anthropocentrée de regles d’association pour l’aide à la décision ». In : *Revue d’Interaction Homme-Machine Vol* 8.2 (2007).

[Cor+05] O. Corcho, M. Fernández-López, A. Gómez-Pérez et A. López-Cima. « Building legal ontologies with METHONTOLOGY and WebODE ». In : *Law and the semantic web*. Springer, 2005, p. 142-157.

[DM06] H. Déjean et J.-L. Meunier. « A System for Converting PDF Documents into Structured XML Format ». en. In : *Document Analysis Systems VII*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, fév. 2006, p. 129-140. doi : 10.1007/11669487_12.

[DWL15] D. Dou, H. Wang et H. Liu. « Semantic data mining : A survey of ontology-based approaches ». In : *Proceedings of the 2015 IEEE 9th international conference on semantic computing (IEEE ICSC 2015)*. IEEE. 2015, p. 244-251.

[Dup11] S. Dupuy-Chessa. « Modélisation en Interaction Homme-Machine et en Système d’Information : à la croisée des chemins ». Habilitation à diriger des recherches, Université de Grenoble. Thèses et habilitations. 2011.

[Est+96] M. Ester, H.-P. Kriegel, J. Sander et X. Xu. « Density-based spatial clustering of applications with noise ». In : *Int. Conf. Knowledge Discovery and Data Mining*. T. 240. 1996, p. 6.

[FPU03] U. M. Fayyad, G. Piatetsky-Shapiro et R. Uthurusamy. « Summary from the KDD-03 panel : data mining : the next 10 years ». In : *ACM Sigkdd Explorations Newsletter* 5.2 (2003), p. 191-196.

[FPS96a] U. Fayyad, G. Piatetsky-Shapiro et P. Smyth. « From data mining to knowledge discovery in databases ». In : *AI magazine* 17.3 (1996), p. 37-37.

[FPS96b] U. Fayyad, G. Piatetsky-Shapiro et P. Smyth. « The KDD process for extracting useful knowledge from volumes of data ». In : *Communications of the ACM* 39.11 (1996), p. 27-34.

[FST03] F. Franěk, W. F. Smyth et Y. Tang. « Computing All Repeats Using Suffix Arrays ». In : *J. Autom. Lang. Comb.* 8.4 (juil. 2003), p. 579-591.

[Fre61] H. Freeman. « On the encoding of arbitrary geometric configurations ». In : *IRE Transactions on Electronic Computers* 2 (1961), p. 260-268.

[Gab08] E. Gabarra. « De la binarisation de documents vers la reconnaissance de symboles dans l'analyse de schémas électriques ». Thèse de doct. Pau, 2008.

[GT05] E. Gabarra et A. Tabbone. « Combining global and local threshold to binarize document of images ». In : *Iberian Conference on Pattern Recognition and Image Analysis*. Springer. 2005, p. 371-378.

[GR12] J. Gantz et D. Reinsel. « The digital universe in 2020 : Big data, bigger digital shadows, and biggest growth in the far east ». In : *IDC iView : IDC Analyze the future* 2007.2012 (2012), p. 1-16.

[Gob+13] M. Gobel, T. Hassan, E. Oro et G. Orsi. « ICDAR 2013 Table Competition ». en. In : IEEE, août 2013, p. 1449-1453. doi : 10.1109/ICDAR.2013.292.

[Gus97] D. Gusfield. « Algorithms on Strings ». In : *Trees and Sequences* (1997), p. 89-180.

[Has09] T. Hassan. « User-guided wrapping of pdf documents using graph matching techniques ». In : *2009 10th International Conference on Document Analysis and Recognition*. IEEE. 2009, p. 631-635.

[HB07] T. Hassan et R. Baumgartner. « Table recognition and understanding from pdf files ». In : *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. T. 2. IEEE. 2007, p. 1143-1147.

[He+16] K. He, X. Zhang, S. Ren et J. Sun. « Deep residual learning for image recognition ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, p. 770-778.

[HSS18] J. Hu, L. Shen et G. Sun. « Squeeze-and-excitation networks ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, p. 7132-7141.

[Huf52] D. A. Huffman. « A method for the construction of minimum-redundancy codes ». In : *Proceedings of the IRE* 40.9 (1952), p. 1098-1101.

[JZJ08] Y.-L. Jia, H.-C. Zhang et Y.-Z. Jing. « A modified Bresenham Algorithm of line Drawing [J] ». In : *Journal of Image and Graphics* 1 (2008).

[Kaa18] I. C. Kaadoud. « Apprentissage de séquences et extraction de règles de réseaux récurrents : application au traçage de schémas techniques. » Thèse de doct. Universié de Bordeaux, 2018.

[KRA17] I. C. Kaadoud, N. Rougier et F. Alexandre. « Implicit knowledge extraction and structuration from electrical diagrams ». In : *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2017, p. 235-241.

[KK16] J. Kärkkäinen et D. Kempa. « Faster external memory LCP array construction ». In : *24th Annual European Symposium on Algorithms (ESA 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2016.

[KR09] J. Kärkkäinen et T. Rantala. « Engineering Radix Sort for Strings ». In : *String Processing and Information Retrieval*. Sous la dir. d'A. Amir, A. Turpin et A. Moffat. Springer Berlin Heidelberg, 2009.

[KS03] J. Kärkkäinen et P. Sanders. « Simple linear work suffix array construction ». In : *International Colloquium on Automata, Languages, and Programming*. Springer. 2003, p. 943-955.

[Kei02] D. A. Keim. « Information Visualization and Visual Data Mining ». In : *IEEE Transactions on Visualization and Computer Graphics* 8.1 (jan. 2002), p. 1-8. doi : 10.1109/2945.981847.

[Kei+08] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer et G. Melançon. « Visual analytics : Definition, process, and challenges ». In : *Information visualization*. Springer, 2008, p. 154-175.

[Kim+03] D. K. Kim, J. S. Sim, H. Park et K. Park. « Linear-time construction of suffix arrays ». In : *Annual Symposium on Combinatorial Pattern Matching*. Springer. 2003, p. 186-199.

[KL15] R. Kitchin et T. P. Lauriault. « Small data in the era of big data ». In : *GeoJournal* 80.4 (2015), p. 463-475.

[KA03] P. Ko et S. Aluru. « Space efficient linear time construction of suffix arrays ». In : *Annual Symposium on Combinatorial Pattern Matching*. Springer. 2003, p. 200-210.

[KSH12] A. Krizhevsky, I. Sutskever et G. E. Hinton. « Imagenet classification with deep convolutional neural networks ». In : *Advances in neural information processing systems*. 2012, p. 1097-1105.

[Kur+01] S. Kurtz, J. V. Choudhuri, E. Ohlebusch, C. Schleiermacher, J. Stoye et R. Giegerich. « REPuter : the manifold applications of repeat analysis on a genomic scale ». In : *Nucleic Acids Research* 29.22 (2001), p. 4633-4642. doi : 10.1093/nar/29.22.4633.

[LeC+98] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner et al. « Gradient-based learning applied to document recognition ». In : *Proceedings of the IEEE* 86.11 (1998), p. 2278-2324.

[Leg04] F. Legrand. « Modélisation de circuits électrotechniques en vue de leur simulation : réalisation d'un simulateur ». Thèse de doct. Bordeaux 1, 2004.

[Leg+03] F. Legrand, N. Couture, B. Renaud et H. Lévi. « Electrical or Electronical diagrams simulation using event driven anlysis ». In : *Proceedings of 4ème conférence Internationale sur l'Automatisation Industrielle*. Sous la dir. d'E. de Technologie Supérieure. Montréal, Canada, juin 2003, cd-rom.

[Leg+04] F. Legrand, H. Levi, N. Couture et J.-J. Charlot. « VHDL-AMS modeling and library building for Power Electrical Engineering ». In : *The 8th IEEE International Workshop on Advanced Motion Control, 2004. AMC'04*. IEEE. 2004, p. 111-116.

[Li+08] G. Li, B. C. Ooi, J. Feng, J. Wang et L. Zhou. « EASE : an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data ». In : *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM. 2008, p. 903-914.

[Lou+17] F. A. Louza, G. P. Telles, S. Hoffmann et C. D. A. Ciferri. « Generalized enhanced suffix array construction in external memory ». In : *Algorithms for Molecular Biology* 12.1 (déc. 2017), p. 26. doi : 10.1186/s13015-017-0117-9.

[LB95] W. S. Lovegrove et D. F. Brailsford. « Document analysis of PDF files : methods, results and implications ». In : *Electronic Publishing–Origination, Dissemination and Design* 8.3 (1995), p. 207-220.

[MM00] J. I. Maletic et A. Marcus. « Data Cleansing : Beyond Integrity Analysis. » In : *Iq*. Citeseer. 2000, p. 200-209.

[MMS+11] N. Malviya, N. Mishra, S. Sahu et al. « Developing university ontology using protégé owl tool : Process and reasoning ». In : *International Journal of Scientific & Engineering Research* 2.9 (2011), p. 1-8.

[McC76] E. M. McCreight. « A space-economical suffix tree construction algorithm ». In : *Journal of the ACM (JACM)* 23.2 (1976), p. 262-272.

[Nar+07] K. Narisawa, S. Inenaga, H. Bannai et M. Takeda. « Efficient computation of substring equivalence classes with suffix arrays ». English. In : *Combinatorial Pattern Matching - 18th Annual Symposium, CPM 2007, Proceedings*. T. 4580 LNCS. 2007, p. 340-351.

[NG08] S. Negash et P. Gray. « Business intelligence ». In : *Handbook on decision support systems 2*. Springer, 2008, p. 175-193.

[ND86] D. A. Norman et S. W. Draper. *User Centered System Design ; New Perspectives on Human-Computer Interaction*. Hillsdale, NJ, USA : L. Erlbaum Associates Inc., 1986.

[Pap73] S. A. Papert. « Uses of technology to enhance education ». In : (1973).

[Pér18a] 1. Péré-Laperne Jacques. « PROCEDE DE RESTRUCTURATION D'OBJETS GRAPHIQUES ». P19475FR00. 18-07-2018.

[Pér18b] J. Péré-Laperne. « (A)KDD for Structuring Destrured Documents ». In : *Proceedings of the 2018 International Conference on Artificial Intelligence ICAI'18*. Las Vegas, NV. 89, USA, juil. 2018.

[Pér18c] J. Péré-Laperne. *1A3I at Consumer Electronics Show (CES)*. Exhibitor, TechWest Sands Expo, stand 50879. Las Vegas, Nevada, États-Unis, États-Unis, https ://www.youtube.com/watch ?v=djTBMABN15k et https ://www.youtube.com/watch ?v=-8Ptk8TiNmA. Jan. 2018.

[Pér19a] J. Péré-Laperne. *1A3I at Consumer Electronics Show (CES)*. Exhibitor, TechWest Sands Expo, stand 50241. Las Vegas, Nevada, États-Unis, https ://www.youtube.com/watch ?v=djTBMABN15k et : https ://www.youtube.com/watch ?v=d52sKX2KFSo. Jan. 2019.

[Pér19b] J. Péré-Laperne. « Identification de symboles dans des documents déstructurés ». In : *Revue des Nouvelles Technologies de l'Information*. Extraction et Gestion des connaissances RNTI-E-35 (2019), p. 297-302.

[PC17] J. Péré-Laperne et N. Couture. « Restructuring Unstructured Documents ». In : *SMART INTERFACES 2017*. Venice, Italy : Berntzen, L. et al., juin 2017, p. 60-65.

[MEL15] G. MELANÇON. « Visualisation d'information ». In : *Techniques de l'ingénieur Gestion de contenus numériques* base documentaire : TIB311DUO.ref. article : h7417 (2015). fre.

[PSY10] S. J. Puglisi, W. F. Smyth et M. Yusufu. « Fast, Practical Algorithms for Computing All the Repeats in a String ». In : *Mathematics in Computer Science* 3.4 (mai 2010), p. 373-389. doi : 10.1007/s11786-010-0033-6.

[PST07] S. J. Puglisi, W. F. Smyth et A. H. Turpin. « A taxonomy of suffix array construction algorithms ». In : *acm Computing Surveys (CSUR)* 39.2 (2007), p. 4.

[Ram+03] J.-Y. Ramel, M. Crucianu, N. Vincent et C. Faure. « Detection, extraction and representation of tables ». In : *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* IEEE. 2003, p. 374-378.

[RI] J.-L. B.-.-.-M. Rigamonti–Denis et L.-.-.-R. Ingold. « XCDF : Un format canonique pour la représentation de documents ». In : ().

[Rig+04] M. Rigamonti, K. Hadjar, D. Lalanne et R. Ingold. « Xed : un outil pour l'extraction et l'analyse de documents PDF ». fr. In : *Conférence Internationale Francophone sur l'Ecrit et le Document (CIFED 04)* (juin 2004).

[Rob01] L. Robadey. « 2 (CREM) ». Thèse de doct. Université de Fribourg, 2001.

[RH11] M. Rønne Jakobsen et K. Hornbæk. « Sizing Up Visualizations : Effects of Display Size in Focus+Context, Overview+Detail, and Zooming Interfaces ». In : *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* CHI '11. Vancouver, BC, Canada : ACM, 2011, p. 1451-1460. doi : 10.1145/1978942.1979156.

[RS17] D. H. Rothchild et S. M. Shieber. « Automatically Determining Versions of Scholarly Articles ». In : (2017).

[Sah+08] S. Saha, S. Bridges, Z. V. Magbanua et D. G. Peterson. « Empirical comparison of ab initio repeat finding programs ». In : *Nucleic Acids Research* 36.7 (2008), p. 2284-2294. doi : 10.1093/nar/gkn064.

[SLB10] C. Salperwyck, V. Lemaire et D. U. d. P. de Bois. « Classification incrémentale supervisée : un panel introductif. » In : *AAFD.* 2010, p. 121-148.

[SGF15] A. M. Sànchez-Riofrò, L. A. Guerras-Martin et F. J. Forcadell. « Business portfolio restructuring : a comprehensive bibliometric review ». In : *Scientometrics* 102.3 (2015), p. 1921-1950.

[Sch17] M. Schubotz. « Augmenting mathematical formulae for more effective querying & efficient presentation ». In : (2017).

[SW15] R. Sedgewick et K. Wayne. *Algorithms, Fourth Edition (Deluxe) : Book and 24-Part Lecture Series.* 1st. Addison-Wesley Professional, 2015.

[SM17] A. O. Shigarov et A. A. Mikhailov. « Rule-based spreadsheet data transformation from arbitrary to relational tables ». In : *Information Systems* 71 (2017), p. 123-136.

[SMA16] A. Shigarov, A. Mikhailov et A. Altaev. « Configurable table structure recognition in untagged PDF documents ». In : *Proceedings of the 2016 ACM Symposium on Document Engineering*. ACM. 2016, p. 119-122.

[Shn96] B. Shneiderman. « The Eyes Have It : A Task by Data Type Taxonomy for Information Visualizations ». In : *Proceedings of the 1996 IEEE Symposium on Visual Languages*. VL '96. Washington, DC, USA : IEEE Computer Society, 1996, p. 336-.

[SZ14] K. Simonyan et A. Zisserman. « Very deep convolutional networks for large-scale image recognition ». In : *arXiv preprint arXiv :1409.1556* (2014).

[SB95] P. N. Smith et D. F. Brailsford. « Towards structured, block-based PDF ». In : *Electronic Publishing–Origination, Dissemination and Design* 8.3 (1995), p. 153-165.

[Stu+19] L. Studer, M. Alberti, V. Pondenkandath, P. Goktepe, T. Kolonko, A. Fischer et al. « A Comprehensive Study of ImageNet Pre-Training for Historical Document Image Analysis ». In : *arXiv preprint arXiv :1905.09113* (2019).

[Sze+17] C. Szegedy, S. Ioffe, V. Vanhoucke et A. A. Alemi. « Inception-v4, inception-resnet and the impact of residual connections on learning ». In : *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[Sze+14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov et al. « Going Deeper with Convolutions ». In : *CoRR* abs/1409.4842 (2014).

[TVF12] T. Taris, V. Vigneras et L. Fadel. « A 900MHz RF energy harvesting module ». In : *10th IEEE International NEWCAS Conference*. IEEE. 2012, p. 445-448.

[Tei+16] I. Teinemaa, M. Dumas, F. M. Maggi et C. Di Francescomarino. « Predictive business process monitoring with structured and unstructured data ». In : *International Conference on Business Process Management*. Springer. 2016, p. 401-417.

[TL03] K. Tombre et B. Lamiroy. « Graphics recognition - from re-engineering to retrieval ». en. In : t. 1. IEEE Comput. Soc, 2003, p. 148-155. doi : 10.1109/ICDAR.2003.1227650.

[Tur+14] V. Turner, J. F. Gantz, D. Reinsel et S. Minton. « The digital universe of opportunities : Rich data and the increasing value of the internet of things ». In : *IDC Analyze the Future* 16 (2014).

[Wei73] P. Weiner. « Linear pattern matching algorithms ». In : *14th Annual Symposium on Switching and Automata Theory (swat 1973)*. IEEE. 1973, p. 1-11.

[YF04] M. Yang et R. Fateman. « Extracting mathematical expressions from postscript documents ». In : *Proceedings of the 2004 international symposium on Symbolic and algebraic computation*. ACM. 2004, p. 305-311.

[Yan+19] Y. Yang, D.-W. Zhou, D.-C. Zhan, H. Xiong et Y. Jiang. « Adaptive Deep Models for Incremental Learning : Considering Capacity Scalability and Sustainability ». In : (2019).

[Yu+19] W. Yu, Z. Li, Q. Zeng et M. Jiang. « Tablepedia : Automating PDF Table Reading in an Experimental Evidence Exploration and Analytic System ». In : (2019).

[YY15] M. Yusufu et G. Yusufu. « Efficient Algorithm for Extracting Complete Repeats from Biological Sequences ». In : *International Journal of Computer Applications* 128.16 (oct. 2015), p. 33-37. doi : 10.5120/ijca2015906752.

[ZB12] R. Zanibbi et D. Blostein. « Recognition and retrieval of mathematical expressions ». In : *International Journal on Document Analysis and Recognition (IJDAR)* 15.4 (2012), p. 331-357.

[Zop+18] B. Zoph, V. Vasudevan, J. Shlens et Q. V. Le. « Learning transferable architectures for scalable image recognition ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, p. 8697-8710.

Glossaire

ActiveX : désigne l'une des technologies du Component Object Model de Microsoft avec COM+ et Distributed COM utilisée en programmation pour permettre le dialogue entre programmes. 4

AutoCad™ : est un logiciel de dessin assisté par ordinateur (DAO) créé en décembre 1982 par AutoDesk™ . 52, 63, 120, 129

AutoDesk™ : est une société d'édition de logiciels, créée en 1982 par John Walker avec douze autres personnes en 1982. Aujourd'hui c'est le 5e plus grand éditeur de logiciels dans le monde. En 2018 son chiffre d'affaires est de 2,2 milliards de dollars. AutoDesk édite le logiciel AutoCad™ . 120, 129

Bitmap : est un mot anglais qui, dans notre cas d'usage, est une image matricielle : image numérique décrite point par point. 121, 143

CAO : Conception Assistée par Ordinateur, c'est une discipline qui permet de modéliser un produit pour en faire une exploitation informatique telle que la simulation ou la création automatique d'informations à partir de la modélisation, l'objectif final étant sa fabrication. Si en DAO un trait est un trait, le même trait en CAO peut représenter une connexion, une canalisation, un fils électrique. iv, vi, 3, 10, 12, 21, 23, 35, 36, 38, 39, 42, 52, 64, 105, 109, 120, 121, 126–128

Caractère : c'est l'association de un ou plusieurs glyphes. On peut ainsi créer un caractère en ajoutant un glyphe d'accent à un glyphe de caractère. Les logiciels informatiques ont accès au dessin de ce glyphe par l'intermédiaire d'une police de caractères : le tracé du glyphe y est le plus souvent défini par un ensemble de points ou de courbes de Bézier qui sont enregistrés dans le fichier PDF en succession de segments de droite ou de polylignes. En informatique, la notion de caractère est une notion qui, dans le principe, associe un nombre à un glyphe, de manière à dissocier la représentation physique du caractère de sa signification avec une police de caractères. 45

DAO : Dessin Assisté par Ordinateur, c'est une discipline qui permet de créer des dessins techniques avec un logiciel informatique. Les dessins produits sont des images vectorielles. iv, vi, 3, 10, 12, 21, 23, 35, 36, 38, 39, 42, 45, 52, 64, 105, 109, 121, 126–128

Elec'View™ : est un logiciel de conception assistée par ordinateur (CAO) créé par la société Algo'Tech Informatique™. 63

Fichier matriciel : c'est un fichier qui contient des images matricielles. Il se différencie en cela des fichiers vectoriels, qui sont constitués d'images vectorielles. On parle aussi de fichier bitmap. 121

Fichier vectoriel : c'est un fichier qui contient des images vectorielles. Il se différencie en cela des fichiers matriciels, qui sont constitués d'images matricielles. v, 121

Glyphe : c'est une représentation graphique (parmi une infinité possible) d'un signe typographique, autrement dit d'un caractère (glyphe de caractère) ou d'un accent (glyphe d'accent). 45, 80–84, 120

Image matricielle : en informatique, c'est une image numérique composée d'une matrice de points colorés. La juxtaposition de ces points colorés forme l'image. Elle se différencie en cela des images vectorielles, qui sont constituées d'objets géométriques. On parle aussi d'image bitmap. 10, 121

Image vectorielle : en informatique, c'est une image numérique composée d'objets géométriques (segments de droite, polygones, coniques ,arcs de conique, courbes de Bézier, etc), avec des attributs (position, couleur, remplissage, visibilité, etc) et auxquels on peut appliquer différentes transformations (homothéties, similitude, rotations, etc). Elle se différencie en cela des images matricielles, qui sont constituées de pixels. 120, 121

Pixel : en informatique, une image numérique composée d'une matrice de points colorés. La juxtaposition de ces points colorés forme l'image. Ces points sont nommés pixels. 16, 126

Scan'Builder™ : est un logiciel de Conception Assistée par Ordinateur (CAO) créé par la société Algo'Tech Informatique™. Il introduit automatiquement les plans papier dans un logiciel de DAO, comme s'ils avaient été saisis manuellement.. 10

See Expert™ : est un logiciel de conception assistée par ordinateur (CAO) créé par la société IGE-XAO™. 63, 108

Thread : c'est une tâche normalisée par ISO/CEI 2382-7:2000 (autres noms possibles : processus léger, fil d'instruction). Il représente l'exécution d'un ensemble d'instructions du langage machine d'un processeur. Les exécutions semblent se dérouler en parallèle. Chaque processus possède sa propre mémoire virtuelle, les threads d'un même processus se partagent sa mémoire virtuelle. Tous les threads possèdent leur propre pile d'exécution. 107

UNICODE : c'est un standard informatique qui permet des échanges de textes dans différentes langues. 80

Acronymes

(A)KDD *Antropocentric Knowledge Discovery in Database*. ii, iii, xiii, 11, 12, 26, 33–35, 37–41, 49, 57, 58, 60, 62, 97, 102, 105, 107, 108

3DS 3D Studio. 142

ADA Langage, dont le nom correspond au prénom d'Ada Lovelace. 134

ADN Acide DésoxyriboNucléique. 27

ASCII *American Standard Code for Information Interchange*. 80

BMP *BitMaP*. 10

CEI Commission Électrotechnique Internationale. 80

CES *Consumer Electronics Show*. 106

CGM *Computer Graphics Metafile*. 142

CIDRE *Coopérative Interactive Document Reverse Engineering*. 22

CIFRE Convention Industrielle de Formation par la REcherche. v, 10, 130

COG *Component Object Graphics*. 21

COMAU *COnsorzio MAcchine Utensilie*. 6, 109

CPU *Central Processing Unit*. 97

DBSCAN *Density-Based Spatial Clustering of Applications with Noise*. 84

DM *Data Minning*. 2

DOLORES *DOcument LOgical REStructuring*. 23

DPI *Dots Per Inch*. 16

DVD *Digital Versatile Disc*. 7, 8

DW *Data Warehouse*. 2

DWF *Design Web Format*. 142

DWG *DraWinG*. 39, 52, 64, 109, 129, 142

DWT *DraWing Template*. 142

DXF *Drawing eXchange Format*. 24, 39, 52, 64, 109, 129, 142

ECD Extractions des Connaissances des Données. 24, 25

ECML-PKDD *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. 106

EGC Extractions et Gestion des Connaissances. 58

eGSA *external memory Generalized Suffix Array*. 29

EIS *Executive Information Systems*. 2

EMF *Enhanced MetaFile*. 52, 64, 142

ERP *Enterprise Requirement Planning*. 2

ETI *Entreprises de Taille Intermédiaire*. 4, 5, 16

FIAT *Fabbrica Italiana Automobili Torino*. 6

GAFA Google, Apple, Facebook et Amazon. 4, 15, 16

Go Giga octets = 1 000 000 000 octets. 97

GPU *Graphics Processing Unit*. 16

HPGL *Hewlett Packard Graphic Language*. 126–128, 142

IADM *Incremental Adaptive Deep Model*. 26

IBM *International Business Machine*. 134, 135

ICAI *International Conference on Applied Informatics*. 106

ICDAR *International Conference on Document Analysis and Recognition*. 24

IDC *International Data Corporation*. 7

IGES *Initial Graphics Exchange Specification*. 142

IHM Interface Homme Machine. 49

ISO *International Organization for Standardization*. 80

JPEG *Joint Photographic Experts Group*. 10

KDD *Knowledge Discovery in Database*. ii, iii, xiii, 11, 12, 24–26, 30, 34, 36–40, 97, 105, 106, 148

LCP *Longest Common Prefix array*. 11, 27–29, 99, 106

LE *Left Extendible*. 145

MIT *Massachusetts Institute of Technology*. 14

NAIA Forum Néo-Aquitain de l'Intelligence Artificielle. 106

NE *No Extendible*. 145

NLE *No Left Extendible*. 148, 150

NRE *No Right Extendible*. 147

OCR *Optical Caracter Recognition*. 45

OLAP *On-Line Analytic Processing*. 2

PAO Publication Assistée par Ordinateur. vi, 3, 12, 21, 23, 35, 36, 38, 42, 52, 64, 126, 128

PC *Personal Computer*. 135

PCL *Printer Command Language*. 142

PCRD Programmes Cadre pour la Recherche et le Développement technologique. 10

PDF *Portable Document File*. ii, iii, v–vii, xiii, xiv, xviii, 1, 4, 8, 11, 12, 14, 17–24, 30–33, 35, 36, 40, 41, 43, 45, 51–56, 80, 81, 85, 103, 105, 106, 127, 128, 137, 138, 142, 143

PID *Process and Instrumentation Diagram*. 9, 109

PME Petites et Moyennes Entreprises. v, 4, 5, 15, 16

PROLOG PROgrammation LOGique. 135

PS *PostScript*. 127, 128, 142

R2E Réalisations et Études Électroniques. 135

RAM *Random Access Memory*. 97

RE *Right Extendible*. 145

SA *Suffixe Array*. 28, 58, 60, 99, 106

SAP *Suffixe Array for Pattern*. xv, 11, 12, 30, 37, 58, 60, 70, 92, 93, 95, 99–102, 107, 145, 149

SATT Sociétés d'Accélération du Transfert de Technologies. 130

SIGKDD *Special Interest Group on Knowledge Discovery and Data Mining*. 25

SLT *StereoLiThography file format (ANSI and binary)*. 142

SOFINS Spécial Opérations Force Innovation Network Séminar. 106

STEP *Standard for The Exchange of Product model data*. 142

SVG et SVGZ *Scalable Vector Graphics*. 142

TIFF *Tagged Image File Format*. 10, 17, 22

TTT *Type by Task Taxonomy of information visualization*. 50

UTF-8 *Universal Character Set Transformation Format - 8 bits*. 80

VRML *Virtual Reality Modeling Language*. 142

WMF *Windows MetaFile*. 142

XCDF *Canonical and Structured Document Format*. 23

Annexes

Annexe A

Pourquoi déstructurer l'information structurée

Pourquoi déstructurer des informations qui sont structurées ? Les trois principales raisons sont décrites ci-dessous.

A.1 La déstructuration pour reproduction de l'information

Dans les logiciels de CAO, DAO, PAO, les informations créées et manipulées sont, en général, très structurées. Pour les visualiser sur l'écran ou pour les imprimer, il faut transformer l'information structurée en information déstructurée. Dans les années 80, il était impossible de demander, à une carte graphique de tracer un carré, un cercle, une lettre vectorielle. La seule possibilité était d'allumer ou d'éteindre les pixels de la carte graphique (en noir et blanc). Pour atteindre ce résultat, tout est décomposé en segments de droite, puis les points du segment de droite sont calculés par l'algorithme de Bresenham de 1965 ([Bre65]), amélioré en 1991 par Angel Edward et Morrison Don ([AM91], puis en 2008, par Jia Yin-Liang, Zhang Huan-Chun et Jing, Ya-Zhi ([JZJ08]. Cette information déstructurée est composée d'objets graphiques très basiques et très simples qui sont compréhensibles par le périphérique utilisé pour visualiser cette information. Pour les écrans ce sont les points de la matrice de l'écran.

A.1.1 La déstructuration pour tracer l'information

Dans les mêmes années, pour les logiciels de CAO/DAO, imprimer des documents consiste à tracer ces documents sur des tables traçantes. Le HPGL (Hewlett Packard Graphic Language, créé en 1977) est le premier protocole d'impression des informations graphiques et textuelles, il a été développé par HP (Hewlett Packard). C'est devenu un standard dans le milieu industriel, tous les éditeurs de logiciels techniques l'utilisent.

Les commandes du HPGL sont simples et peu nombreuses, mais le gros avantage c'est quelles traçent directement les segments de droite qui avaient été calculés pour l'affichage sur l'écran. Cette correspondance permet de réutiliser les mêmes routines de déstructuration de l'information en segments de droite. Pour retranscrire l'information graphique et textuelle, on n'utilise que cinq commandes de base : « Sélectionner Plume » (exemple : SP1), « Aller à la Position X Y » (exemple : PA 100,100), « Plume Haute »(exemple : PU), « Plume Basse » (exemple : PD), et « Changer de feuille » (exemple : EC). Avec ces 5 commandes, la retranscription de l'image qui s'affiche sur l'écran est exactement la même que celle tracée par les tables traçantes (puisqu'on utilise les mêmes segments de droite). Les utilisateurs n'acceptent pas que l'image affichée à l'écran soit différente de l'image tracée par le traceur. Par exemple, il faut que les segments de droite qui forment les lignes en traits mixtes soient exactement de la même longueur et à la même position sur l'écran que sur la table traçante.

Un trait discontinu ést décomposé en autant de segments de droite qu'il contient de petits traits. De même les cercles, les arcs de cercles, les ellipses ou les arcs d'ellipses, toutes ces coniques, mais aussi les carrés, les rectangles, les parallélogrammes sont transformés en petits segments. Les remplissages et les hachures subissent la même décomposition en segments de droite, en tenant compte de l'épaisseur de la plume afin que les remplissages soient complets.

Les textes subissent aussi cette transformation, chaque lettre est décomposée en polylignes puis remplie. Les polylignes et les remplissages sont à leur tour décomposés en segments de droite. On utilise rarement les polices présentes sur le traceur car il faut que le rendu sur la table traçante et à l'écran soit exactement le même. Et comme nous ne connaissons pas le tracé précis de la lettre dans le traceur nous préférons utiliser nos polices de caractères internes.

A la différence de l'affichage à l'écran qui se fait directement par le programme, pour les traceurs, un fichier est créé pour être repris par l'ordinateur qui pilote le traceur (il n'y a souvent qu'un traceur et les réseaux étaient très peu développés). Donc, l'ensemble des commandes, nécessaire au traçage du document, est enregistré dans un fichier. De cette façon, il est possible de retracer le document sans avoir besoin du logiciel. Ce fichier peut être transmis ou communiqué à une autre personne.

A.1.2 La déstructuration pour imprimer l'information

Puis sont arrivés les formats PS (PostScript™, créé en 1982 par Adobe™), et PDF (Portable Document Format, créé en 1992 par Adobe™ pour remplacer le PS). Ces formats ont permis d'imprimer les graphiques des logiciels de DAO/CAO sur des imprimantes laser, mais aussi sur de très nombreux périphériques car les constructeurs de ces périphériques proposent des interpréteurs du format de fichier PDF.

Bien que les formats PDF et PS, qui reposent sur des langages à pile, per-

mettent d'enregistrer les informations de façon très structurée, cette possibilité n'a jamais (ou presque) été utilisée par les éditeurs des logiciels DAO/CAO ou de PAO (Publication Assistée par Ordinateur). Dans les fichiers PDF, on retrouve la même déstructuration de l'information que pour les traceurs, ceci pour les deux raisons suivantes.

La première : quand les formats PS et PDF sont arrivés, les éditeurs de logiciels ont utilisé les mêmes routines d'édition, pour créer ces types de fichiers que celles qui avaient été développées pour l'édition sur les traceurs. Car il a été très simple de transformer les ordres d'édition pour les traceurs en ordres d'édition dans ces fichiers. Les mêmes ordres d'édition existaient des 2 côtés, seul le nom et la syntaxe de la commande était différents. Pour les textes, les évolutions du matériel ont permis d'avoir les mêmes polices de caractères à l'écran et dans les fichiers PDF. Dans les PDF récents, les lettres sont rarement décomposées en segments de droite (mais cela peut arriver) car les écrans et les imprimantes disposent, exactement, des mêmes polices de caractères.

La deuxième : c'est qu'avec des informations déstructurées, il est plus difficile de faire du reverse engineering sur ces fichiers contenant du PDF. Ils apparaissent comme des fichiers verrouillés, ce qui n'est pas du tout le cas.

A.2 La déstructuration pour archivage

L'archivage des fichiers informatiques (HPGL ou PDF) qui permettent l'impression des informations sur les traceurs à plumes ou sur les imprimantes laser, s'est développé. Cet archivage facilite la réédition des documents sans avoir besoin du logiciel ni des informations structurées qui sont présentes dans le logiciel. Cet archivage complète les sauvegardes faites pour conserver les informations structurées des logiciels.

A.3 La déstructuration à la demande des clients

Les utilisateurs des logiciels de DAO/CAO souhaitent disposer de fichiers, contenant des schémas ou des plans, qu'ils peuvent transmettre à leurs partenaires, clients ou fournisseurs, sans « diffuser leur savoir-faire ». C'est pour cette raison que les fichiers, pour les traceurs ou les fichiers PDF, ont été archivés et transmis à l'occasion, en toute confiance, à leur partenaire, comme ils l'auraient fait avec des plans sur papier. Les plans papier sont remplacés par ces fichiers.

Ces fichiers facilitent la transmission de l'information graphique sans transmettre le savoir-faire de la société qui a conçu ces plans. La personne qui reçoit un fichier PDF ne peut pas modifier l'information contenue dans le fichier. Elle ne peut pas l'extraire ou la réutiliser. Elle ne peut que la visualiser et l'imprimer. Ceci est vrai et faux. Faux car il est possible de relire et donc d'extraire l'information contenue dans ces fichiers, et vrai car, même si on accède à cette information, comme elle est très déstructurée, elle n'est pas exploitable par des logiciels.

Il faut comprendre que cette demande des utilisateurs est très forte. Par exemple, pour d'autres types de fichiers vectoriels que l'on peut générer, comme les fichiers DXF (*Drawing eXchange Format*) ou DWG (*DraWinG*) qui sont des formats de fichiers développés par AutoDesk™ pour son logiciel AutoCad™, les éditeurs de logiciels ont une option qui permet de créer un fichier structuré ou déstructuré. Le fait d'enregistrer des informations déstructurées dans ces fichiers rend plus difficilement exploitable, et même quasiment inexploitable, le contenu de ces fichiers, mais le rendu visuel est parfait. Le savoir-faire du client est ainsi protégé, ainsi que le savoir-faire de l'éditeur du logiciel.

Annexe B

Pourquoi j'ai fait une thèse ?

J'aurais pu confier ces travaux de recherche directement à un laboratoire, mais, mes expériences dans le cadre des projets de recherche et des thèses CIFRE que j'ai menés, m'ont montré que les objectifs des laboratoires et ceux des industriels étaient différents.

L'objectif d'un chercheur est rarement d'arriver à un produit industrialisable. Dès qu'il a trouvé ou solutionné un problème ou une difficulté, il passe à une autre difficulté ou à un autre problème, il cherche dans une autre direction.

Ceci est une démarche normale pour un chercheur, pas pour un industriel. Pour passer de la solution élaborée dans le cadre des travaux de recherche à son application industrielle, c'est très complexe, et très difficile, et je le dis en connaissance de cause, en m'appuyant sur plus de 10 projets de recherche et 7 thèses.

Si la recherche solutionne le ou les problèmes les plus importants d'une façon théorique, le passage à son industrialisation nécessite souvent de reprendre les principes de base. Leur application industrielle, compte-tenu du contexte industriel, pour arriver à une solution viable pour le client final, est impossible sans modification et adaptation. Cette phase d'industrialisation est fondamentale. Les outils utilisés en laboratoire pour prototyper une solution sont rarement utilisables en l'état en milieu industriel. Il est nécessaire de les changer, ce qui peut avoir des répercussions très importantes sur les méthodes développées et demande souvent de les modifier ou de les adapter.

La mise en place des Sociétés d'Accélération du Transfert de Technologies (SATT) par un ou plusieurs établissements (universités et organismes de recherche), pour détecter et évaluer les inventions issues de laboratoires de la recherche publique et les accompagner jusqu'à leur transfert vers des entreprises, est une des réponses proposées par le Ministère de l'Enseignement Supérieur de la Recherche et de l'Innovation. Pour modifier ou adapter ces travaux de base, il faut les maîtriser parfaitement. C'est le cas du chercheur et du laboratoire, et rarement celui de l'industriel.

On peut comparer un laboratoire à une mine de pierres précieuses : le chercheur cherche les filons, met à nu un certain nombre de pierres précieuses (mais pas toutes) parfois il peut en extraire quelques-unes mais c'est très rare, et,

encore plus rarement, il les taillera ou les cisèlera pour en faire des pierres précieuses directement exploitables.

Le fait pour l'industriel d'aller dans la mine, de prendre les outils adaptés pour extraire ces pierres, puis de les tailler et de les ciseler, est très difficile, même impossible s'il ne maîtrise pas parfaitement le sujet. C'est pourquoi j'ai fait une thèse, pour maîtriser la totalité du sujet, et du processus, de la recherche à son industrialisation.

Annexe C

Évolution du document au cours des âges

C.1 Les documents jusqu'au 19e siècle

. Quelques dates significatives :
— Les plus vieux fossiles du genre « Homo » remontent à 2,8 millions d'années
— Les plus vieux outils (le Chopper) auraient été utilisés il y a 2,3 millions d'années
— Les premiers « Homo sapiens » seraient arrivés il y a 200 millions d'années
— Les premiers outils composites silex plus bois sont arrivés entre 200 000 ans et 40 000 ans
— Les premiers outils composés d'os d'animaux taillés par l'homme, remontent à environ 30 000 ans
— Les peintures rupestres sont arrivées il y a 26 000 ans sur à peu près tous les continents, les peintures des grottes de Lascaux remontent à 26 000 ans avant notre ère
— Les archéologues positionnent les sceaux chinois qui servaient, semble-t-il, à la commercialisation des denrées à environ 7 000 ans avant notre ère
— Les premières tablettes d'argile ont été fabriquées par l'homme en Mésopotamie ; elles contenaient des hiéroglyphes qui servaient essentiellement à la commercialisation des biens, puis à la définition des lois, ceci environ 3000 ans avant notre ère
— Légèrement un peu plus tard, c'est le papyrus qui a été utilisé par les égyptiens pour communiquer, puis le parchemin
— Enfin le papier est découvert par les chinois 100 ans avant notre ère.

Depuis cette découverte et surtout depuis l'invention de l'imprimerie par Gutenberg en 1450, c'est le papier qui est le support le plus utilisé pour transmettre une information. Le papier (voir figure C.1, page 133) a remplacé la transmission orale de l'information, et, il a complété celle des bas-reliefs, des sculptures ou des peintures sur bois et pierre. Au milieu du 20e siècle, c'est le papier qui

représentait la majorité des documents. À côté du papier et des calques, on peut ajouter d'autres supports comme, les microfiches, les photos, les films, les enregistrements sonores, les peintures et les sculptures. Sur une échelle logarithmique cela donne :

— -2 000 000 années : apparition de l'Homo
— -200 000 années : apparition de l'Homo sapiens
— -20 000 années : premières peintures et outils
— -2 000 années : papyrus et parchemin
— -200 années : le papier.

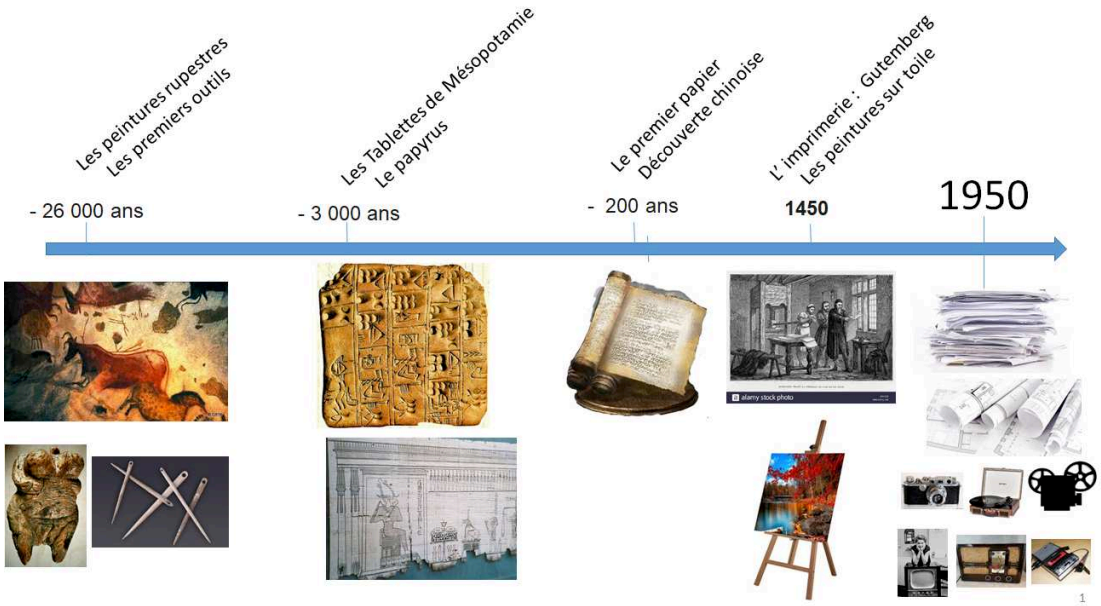


FIGURE C.1 – Évolution du document au cours des âges

Si l'on reprend la définition d'un document, un support avec une information, il faut un moyen pour transcrire cette information sur le support, et, bien sûr, un moyen pour interpréter cette information. On peut considérer que jusqu'au début du 20e siècle, le moyen pour transcrire cette information, c'était l'homme et le moyen pour interpréter cette information c'était encore l'homme en utilisant ses cinq sens et plus particulièrement la vue, l'ouïe et le toucher.

Au cours des 2 derniers millénaires, c'est l'écriture qui s'est imposée comme moyen de transcription de l'information : au début, sur des papyrus, puis sur le parchemin et enfin sur le papier. Le papier est devenu le support le plus utilisé jusqu'au milieu de 20e siècle.

C.2 Les documents depuis le 19e siècle

En 2020, 99% de l'information sera numérique. Comment s'est faite cette évolution ? Cette transformation s'est essentiellement produite pendant ces 50 dernières années, mais elle a été initiée au 19e siècle, sur des inventions du 18e siècle .

C.2.1 Le 19e siècle

En 1801, les métiers à tisser JACQUARD, mis au point par Marie Joseph Jacquard, à Lyon, fonctionnaient avec des cartes perforées en carton. Ces cartes perforées sur carton ont été inventées par Jean-Baptiste Falcon en 1728. Il a remplacé le papier des rubans perforés inventés par Basile Bouchon en 1725, par du carton. Ces rubans étaient utilisés pour piloter les premiers métiers à tisser.

Les cartes perforées ont été également utilisées pour faire fonctionner des automates, les orgues de barbarie et les pianos mécaniques.

De 1834 à 1910, Charles Babbage, puis son fils Henry, vont développer les premières machines à calculer analytiques puis différentielles. En incorporant ces cartes perforées dans les machines, ils leur donnent le premier périphérique d'entrée qui peut automatiser les actions. Par le développement de ces machines, ils mettent en place toutes les notions que l'on retrouve, encore aujourd'hui, dans les ordinateurs, l'unité centrale, la mémoire de stockage, et les périphériques d'entrée et de sortie.

Henry Babbage arrêta le développement de la machine en 1888, après une erreur de calcul lors du calcul et de l'impression de la table des quarante premiers multiples de Π, avec vingt-neuf décimales. Il reprit le développement en 1906 et donna une démonstration réussie de ce calcul devant l'académie royale anglaise d'astronomie.

Pour rappel, la première machine à calculer a été inventée par Pascal, en 1642, la Pascaline. Elle permettait de faire des additions et des soustractions.

C'est au cours du développement des machines de Babbage que Ada Lovelace formalise les idées de Babbage et développe le premier algorithme de programmation de l'histoire. Elle est la première informaticienne de l'humanité. Elle s'appuie sur un véritable algorithme très détaillé pour calculer les nombres de Bernoulli. Le programme qui en résulte est souvent considéré comme le premier véritable programme informatique au monde.

Mais plus qu'une machine à calculer elle voit dans la machine un calculateur universel : « Beaucoup de personnes [...] s'imaginent que parce que la Machine fournit des résultats sous une forme numérique, alors la nature de ses processus doit être forcément arithmétique et numérique, plutôt qu'algébrique ou analytique. Ceci est une erreur. La machine peut arranger et combiner les quantités numériques exactement comme si elles étaient des lettres, ou tout autre symbole général ; en fait, elle peut donner des résultats en notation algébrique, avec des conventions appropriées. ». Ada Lovelace est peu connue en France sauf par les programmeurs qui ont entendu parler du langage ADA, nommé en son honneur. C'est le début de l'informatique et de la programmation.

Et cela ne fait que s'accélérer de façon exponentielle. En 1890 : Hollerith commercialise des machines à calculer électriques qui traitent automatiquement les données d'un recensement aux États-Unis en 1896 : il crée une société appelée « Tabulation Machine Corporation », qui, en 1924, devient « International Business Machine » (IBM™), qui existe toujours.

C.2.2 Le 20e siècle

En 1936 : Alan Turing propose sa définition des « machines de Turing », pendant la guerre de 1939 à 1945, il conçoit une machine pour décrypter les messages allemands, ce qui contribuera à la victoire des alliés.

Et en 1945 : John Von Neumann propose l'architecture d'un calculateur universel (c'est la naissance de l'ordinateur), appelée architecture de Von Neumann ». Le premier ordinateur, en 1946, fait 30 tonnes, occupe 160 m2, il utilise comme mémoire des tubes à vide, et il a la puissance d'une petite calculatrice.

1958 : l'IBM 7044, est le premier ordinateur intégrant des transistors, il a 64 Koctets de mémoire.

En 1973 : R2E™, société française, crée le premier micro-ordinateur : le Micral™. Il est muni d'un clavier et d'un écran. Ils utilise le langage PROLOG™ (PROgrammation LOGique). C'est le premier micro-ordinateur que j'ai utilisé (en 1976) pour le Ministère des Finances, et sur lequel j'ai développé, avec mes équipes, les premiers applicatifs (consolidation des plans comptables des compagnies d'assurances).

En 1981 : l'IBM PC™, (Personal Computer - modèle 5150) fait son apparition avec le Pascal comme langage de programmation.

En 1984 : Le McIntosh™ d'Apple™ utilise les interfaces graphiques et la souris ; les premiers langages objets sont développés.

Le monde du tout numérique est en route.

C.2.3 L'impact sur la production des documents

C'est à partir de là que la production des documents va évoluer. Pendant 2000 ans, c'est l'homme qui produit directement les documents en se servant manuellement d'outils, et c'est l'homme qui interprète directement les documents avec l'utilisation de ses 5 sens. Avec l'arrivée de l'ordinateur, c'est encore l'homme qui crée l'information mais elle est enregistrée et stockée sur des supports non directement interprétables par l'homme. L'information est numérique et elle est stockée sur des supports qui ne sont exploitables par l'homme que grâce à un périphérique de lecture (voir figure C.2, page 136).

Ceci pose un énorme problème d'exploitation et de conservation de cette information, sans périphérique de lecture, il n'y a pas d'exploitation possible de l'information. Et souvent, sans le logiciel qui a produit l'information, il n'y a plus la possibilité d'exploiter l'information de la même façon.

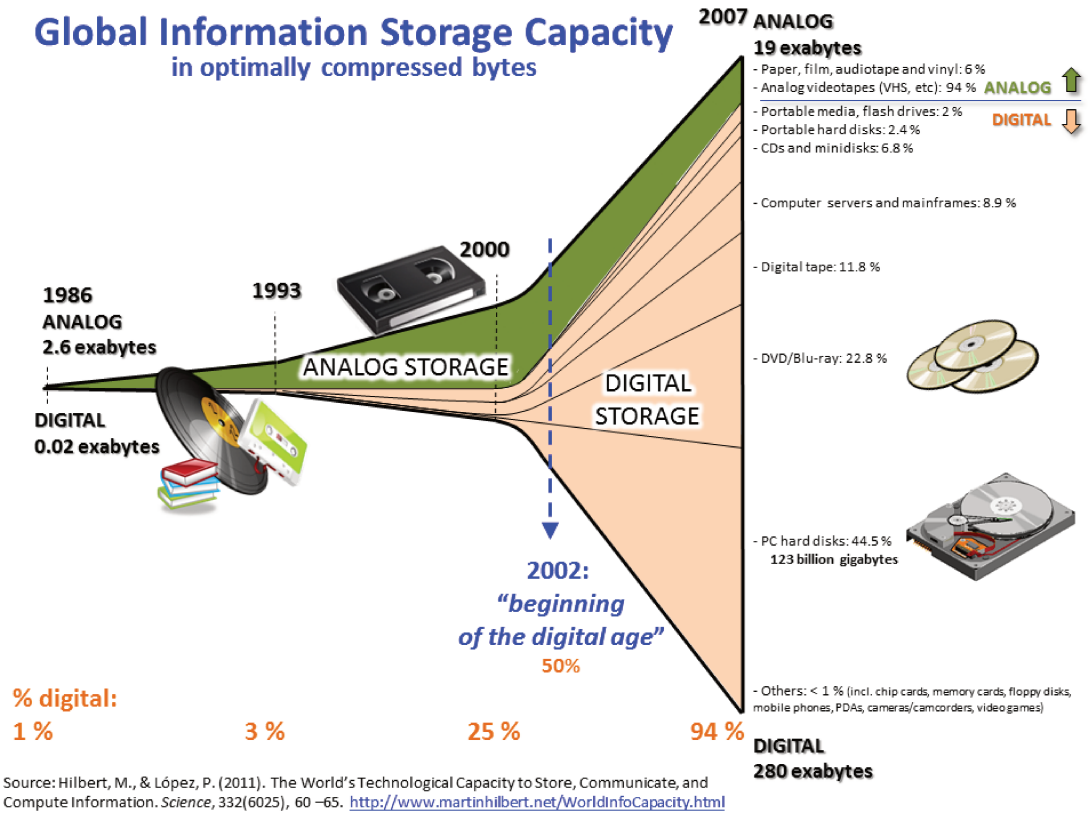


FIGURE C.2 – Ventilation des volumes de stockage de l'information par type de support

Annexe D

Exemples de documents à restructurer

D.1 Structure d'un dossier électrique

Prenons un premier exemple dans le domaine de la schématique électrique (voir figure D.1) avec un dossier d'automatisme ou d'électromécanique archivé dans un fichier PDF. Le fichier d'archivage de ce schéma, au format PDF, ne contient que des segments de droite, des poly lignes, des choix de stylo (couleur, épaisseur), des lettres isolées, le mot « KM34.1 » n'existe pas, on trouve la lettre « K » à une certaine position, puis la lettre « M » à une autre position, et ainsi

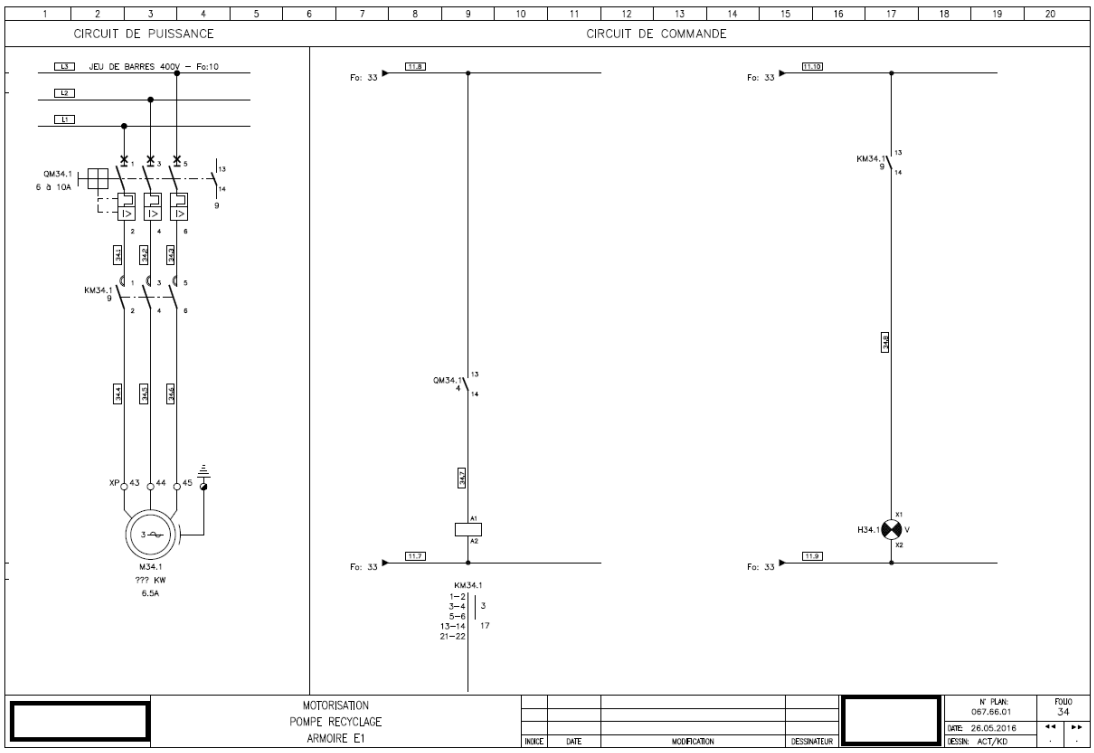
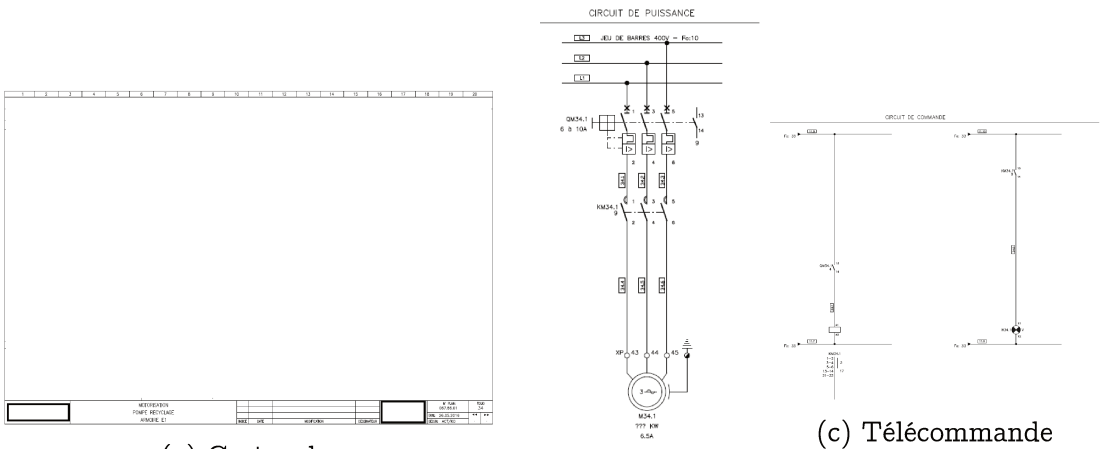


FIGURE D.1 – Exemple de schéma électrique circuit puissance et télécommande d'un démarrage moteur



(a) Cartouche

(b) Puissance

(c) Télécommande

FIGURE D.2 – Identification des 3 parties principales du folio : le cartouche, la puissance et la télécommande

de suite pour les autres lettres ou chiffres. Les cercles et les remplissages sont également décomposés en segments de droite.

Cet ensemble d'objets « basiques » fait que le document est non structuré par rapport à la vision d'un homme de l'art qui, lui, a une interprétation très structurée du document quand il l'analyse visuellement.

Ce document déstructuré au format PDF contient des pages, qui sont appelées des folios et l'ensemble de ces folios forme le dossier électrique.

D.1.1 Structure du folio

Dans la figure D.1, page 137, nous avons choisi un des folios du dossier qui représente un démarrage moteur simple avec son circuit puissance et son circuit de télécommande.

Dans ce schéma, une personne de l'art connaissant les métiers de l'électricité et de l'automatisme va identifier (voir figure D.2) :

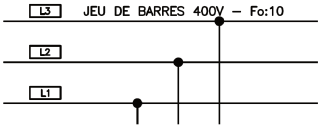
— le « cartouche »
— le circuit puissance
— le circuit télécommande

et comprendre que ce schéma est une partie d'un schéma d'automatisme.

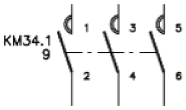
D.1.2 La partie puissance

Sur le circuit puissance, il identifie (voir figure D.3, page 139) :

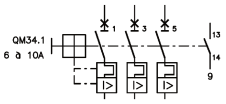
— les fils de puissance (L1, L2, L3)
— la protection du moteur (Q34.1)
— le sectionnement du moteur (KM34.1)
— le moteur (M34.1), avec les bornes 43, 44 et 45 du bornier XP



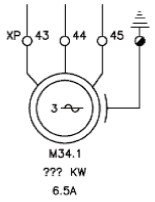
(a) Fils de puissance



(b) Sectionneur



(c) Protection

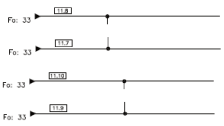


(d) Moteur

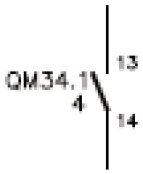
FIGURE D.3 – Identification des symboles du circuit de puissance : les fils, le sectionneur, la protection le moteur

— les équipotentielles 34.1, 34.2, 34.3
— les équipotentielles 34.4, 34.5, 34.6.

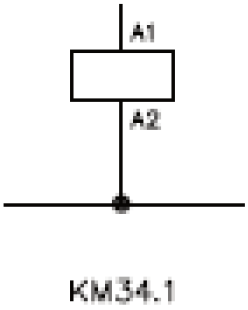
D.1.3 La partie télécommande



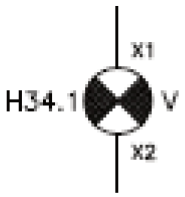
(a) Fils de commande



(b) Pré-coupure



(c) Bobine



(d) Voyant

FIGURE D.4 – Identification des symboles du circuit de télécommande : les fils, la pré-coupure, la bobine et le voyant

Sur le circuit télécommande (voir figure D.2, page 138 et figure D.4), il identifie :

— une partie relayage avec :
  — les fils d'alimentation de la bobine (11.8 et 11.7),
  — le contact à fermeture de la protection du moteur (QM34.1),
  — la bobine du sectionnement du moteur (KM34.1) avec la position des contacts de puissance, des contacts à ouverture et fermeture de la bobine.
— une partie signalisation avec :
  — les fils d'alimentation du voyant de signalisation (11.10, 11.9),
  — le contact à fermeture de la bobine (KM34.1)
  — le voyant de signalisation (H34.1).

La démarche d'un expert pour interpréter un schéma électrique est une démarche descendante (*top-down*). Il part du dossier dans son ensemble. Dans ce dossier il identifie les folios qui correspondent à des fonctions (dans notre cas le démarrage du moteur« KM34.1 »). Puis, il identifie les deux principales sous-fonctions (le circuit de puissance du moteur et le circuit de télécommande). A l'intérieur de ces sous fonctions, il va identifier très facilement les symboles

(le sectionneur, la protection, etc...) les équipotentielles (fils de puissance et de télécommande). Il sait associer un ensemble de segments de droite ou de formes géométriques pour dire que cet ensemble graphique est un symbole qui représente la protection. Enfin, il va savoir lier les textes aux symboles et aux équipotentielles et comprendre, par exemple, que la bobine de la partie télécommande a pour repère « KM34.1 ».

La restructuration va consister obtenir le même résultat que celui de l'expert, mais en utilisant une démarche ascendante (*bottom-up*) pour identifier les formes et les lettres afin d'obtenir des symboles et des textes, lier les textes aux symboles, identifier les liaisons (équipotentielles) connectées à ces symboles. Une fois que ces symboles et ces liaisons (équipotentielles) sont identifiés, nous aurons une démarche descendante, analogue à celle de l'expert pour restructurer le dossier, les fonctions et les sous-fonctions.

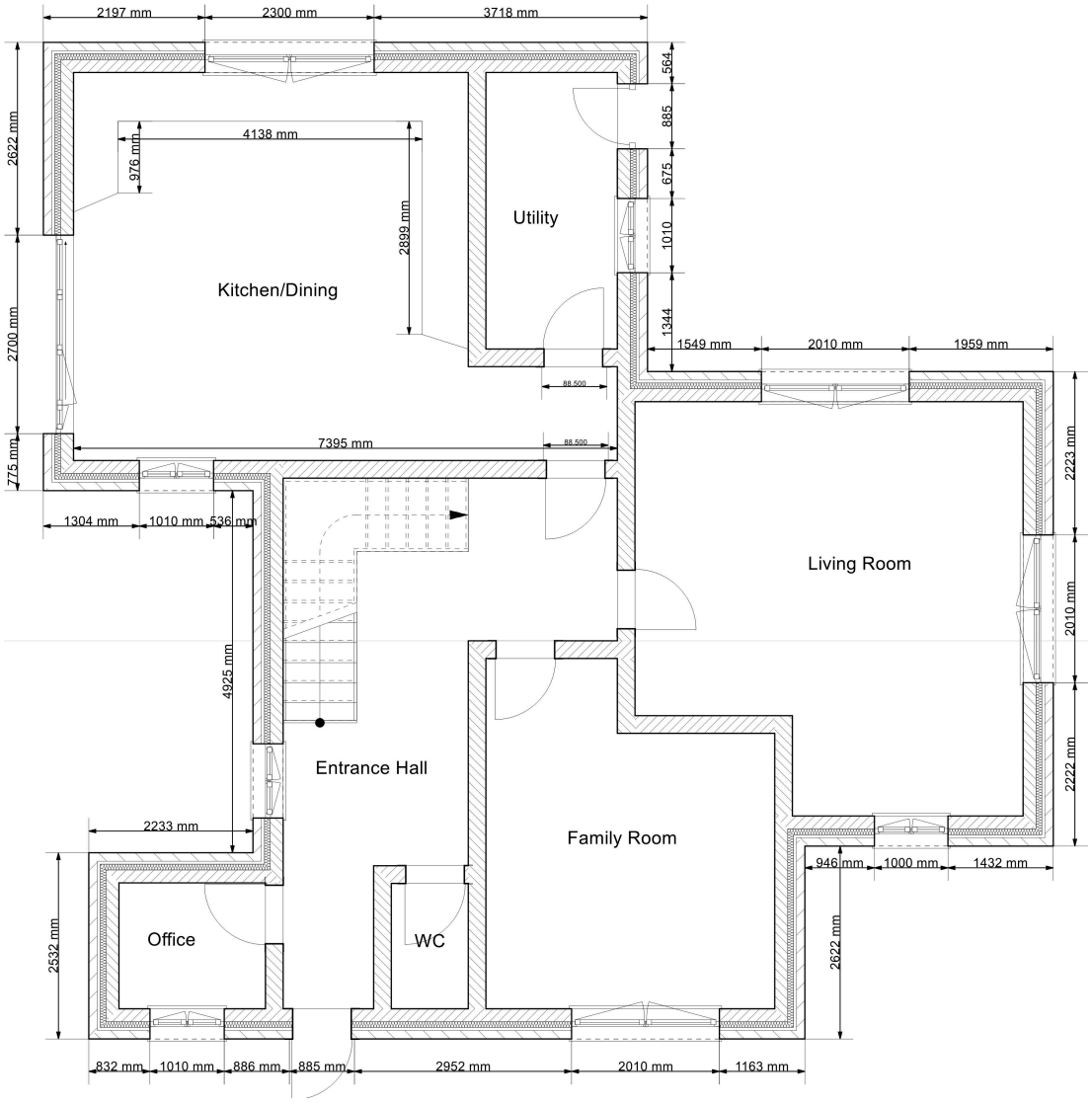


FIGURE D.5 – Exemple 1 : Plan de bâtiment à restructurer

D.2 Autres exemples de documents à restructurer

Deux des exemples suivants sont dans d'autres domaines que ceux de la schématique électrique, le bâtiment (voir figure D.5, page 140) et la mécanique (voir figure D.6). Le dernier exemple est du domaine de la schématique électrique (voir figure D.7, page 142).

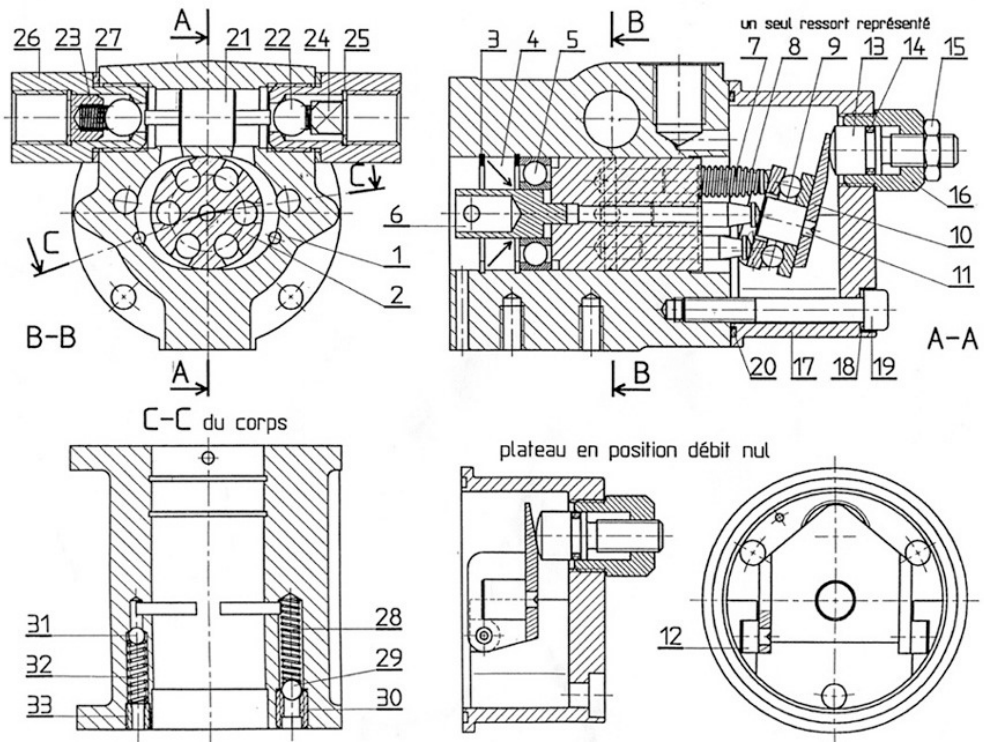


FIGURE D.6 – Exemple 2 : Plan mécanique à restructurer

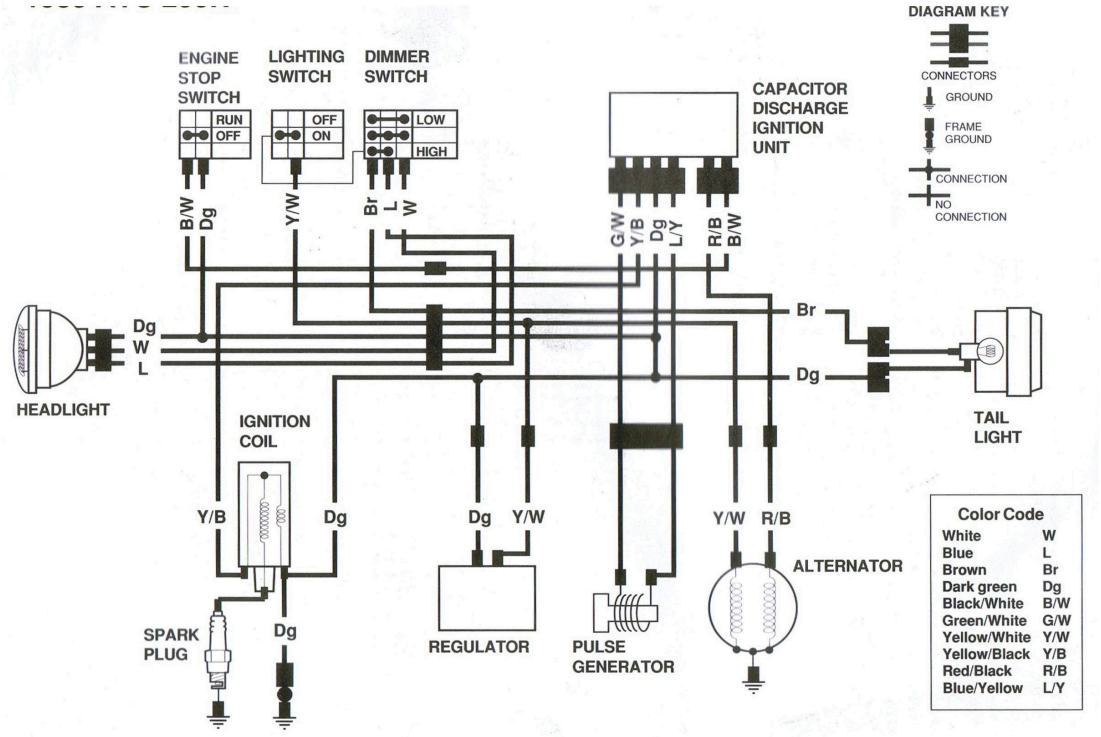


FIGURE D.7 – Exemple 3 : Plan électrique à restructurer

D.3 Type de documents traités

Ces documents graphiques déstructurés traités par le module de lecture sont :

- ***P**ortable **D**ocument **F**ile* (PDF)
- ***P**ost**S**cript* (PS)
- ***S**calable **V**ector **G**raphics* (SVG et SVGZ)
- ***E**nhanced **M**eta**F**ile* (EMF)
- ***W**indows **M**eta**F**ile* (WMF)
- ***D**rawing e**X**change **F**ormat* (DXF)
- ***H**ewlett **P**ackard **G**raphic **L**anguage* (HPGL) et aussi HGL, HG, HPG, PLO, HP, HP1, HP2, HP3, HPGL, HPGL2, HPP, GL, GL2, PRN, SPL, RTL
- ***P**rinter **C**ommand **L**anguage* (PCL) et PLT
- ***DraWinG*** (DWG)
- ***DraWing T**emplate* (DWT)
- ***D**esign **W**eb **F**ormat* (DWF)
- ***C**omputer **G**raphics **M**etafile* (CGM)
- ***I**nitial **G**raphics **E**xchange **S**pecification* (IGES)
- ***S**tandard for **T**he **E**xchange of **P**roduct model data* (STEP)
- **3D** Studio (3DS) et beaucoup d'autres
- ***V**irtual **R**eality **M**odeling **L**anguage* (VRML)
- ***StereoLiT**hography file format (ANSI and binary)* (SLT)

Cette liste n'est pas exhaustive mais elle reprend les principaux formats

de fichiers que nous traitons dans le cadre de cette thèse. Les 10 premiers formats représentent à eux seul plus de 90% du volume des données vectorielles déstructurées présentes dans le milieu industriel.

La prise en charge de ces formats de fichiers est plus un travail d'ingénierie qu'un travail de recherche. C'est le travail que nous avons fait pendant les 2 ans de maturation de notre projet de thèse. Nous verrons que le travail de lecture directe de tous ces fichiers est important, en particulier l'interprétation des fichiers PDF, car elle nous permet d'aborder le problème de la restructuration, sous un nouvel angle. Nous faisons la différence entre les fichiers de type vectoriel et les fichiers de type image (*bitmap*) et cette différence est fondamentale.

Annexe E

La méthode SAP détaillée

E.1 Les notions de base

Nous allons définir les notions de base pour comprendre la notion de tableau des suffixes. Nous utilisons les notations habituelles des chaînes qui sont d'ailleurs utilisées par Puglisi [PSY10] .

L'ensemble Σ désigne un alphabet de codes. Une chaîne S est une séquence ordonnée de codes de Σ. Cette chaîne S est matérialisée par le tableau $S[0..n-1]$. Le dernier objet de la chaîne $S[n-1]$ doit être inférieur à tous les codes de l'alphabet présents dans la chaîne. La longueur de la chaîne $n = |S|$ est $n \geq 1$.

Toute paire d'entiers i et j telle que $0 \leq i \leq j \leq n-1$, permet de définir une sous-chaîne $S[i..j]$ de S tel que $S[i..j] = S[i]S[i+1]...S[j]$. On dit que $S[i..j]$ commence à la position i de S , qu'elle se termine à la position j de S et qu'elle a la longueur $j - i + 1$.

En particulier, $S[0, i]$ est le préfixe $(pref_i(S))$ de S qui se termine à la position i.

$S[i, n-1]$ est le suffixe $(suff_i(S))$ de S qui commence à la position i.

$pref(S)$ et $suff(S)$ dénote l'ensemble de tous les préfixes et suffixes non vides de S.

De la même manière $S[i..j]$ est le préfixe du suffixe $S[i, n-1]$ de longueur $j - i + 1$.

Une répétition est un ensemble de sous-chaînes répétées. Une répétition dans S est définie comme un tuple $R_{S,u} = (p; i_1, i_2, ..., i_e)$, où $e \geq 2$ et $0 \leq i_1 < i_2 < ... < i_e \leq n-1$.

Dans la sous-chaîne qui se répète, u est le générateur, $u = S[i_1..i_1 + p - 1] = S[i_2..i_2 + p - 1] = ... = S[i_e..i_e + p - 1]$, p la période (la longueur), e l'exposant qui est égal au nombre de répétitions, et $i_1, i_2, ..., i_e$ sont les positions des occurrences de $R_{S,u}$, u est le préfixe de longueur p des suffixes $S[i_1..n], S[i_2..n], ..., S[i_e..n]$.

Prenons un exemple, dans la fig E.1, page 145, la sous-chaîne « ISS » a pour période $p = 3$ et $(i_1 = 1)$, 1 est la première position de la sous-chaîne « ISS »,

FIGURE E.1 – Le mot « MISSISSIPPI »

et ($i_2 = 4$), 4 est la deuxième position de la chaîne « ISS » dans la chaîne « MISSISSIPPI ».

Les sous chaînes peuvent être adjacentes si $j \in 1..e-1$, et $i_{j+1} - i_j = p$. Les sous chaînes peuvent se chevaucher si $j \in 1..e-1$, et $i_{j+1} - i_j < p$. Nous disons que $R_{S,u}$ est complète si pour chaque $i \in 0..n-1$ et $i \ni (i_1, i_2, ..., i_e)$, nous avons $S[i..i+p-1] \neq u$.

On dit que $R_{S,u}$ est extensible à gauche (*Left Extendible* - LE) si et seulement si la répétition $(p+1; i_1 - 1, i_2 - 1, ..., i_e - 1)$ existe. Dans ce cas, $(p+1; i_1 - 1, i_2 - 1, ..., i_e - 1)$ est une répétition dont les suffixes de longueur p sont spécifiés par $R_{S,u}$.

De même, $R_{S,u}$ est extensible à droite (*Right Extendible* - RE) si, et seulement si, la répétition $(p+1; i_1, i_2, ..., i_e)$ existe. Dans ce cas $(p+1; i_1, i_2, ..., i_e)$ est une répétition dont les préfixes de longueur p sont spécifiés p $R_{S,u}$.

Si $R_{S,u}$ n'est ni LE ni RE, on dit qu'il est non extensible (*Not Extendible* - NE).

Sur la figure E.1 il y a plusieurs répétitions. Par exemple $R_{S,ISSI} = (4; 1, 4)$ est une répétition non extensible (NE) , tandis que $R_{S,SSI} = (3; 2, 5)$ est extensible à gauche (LE) et $R_{S,ISS} = (3; 1, 4)$ est extensible à droite (RE). Toutes ces répétitions sont complètes. On constate que dans $R_{S,ISSI}$ les sous chaînes se chevauchent (lettre « I »).

E.2 Notre méthode : SAP (*Suffixe Array for Pattern*)

De nombreuses solutions pour trouver les répétitions utilisent le tableau des suffixes (ou l'arbre des suffixes), puis le tableau des plus longs préfixes communs, ainsi que d'autres tableaux. A partir de ces tableaux, ils développent un algorithme qui permet d'identifier les répétitions.

Dans notre algorithme « SAP » nous partons directement de la chaîne de départ S. Nous trions une partie de l'ensemble des suffixes de la chaîne que nous stockons dans une structure proche du tableau des suffixes. Pour le tri

des suffixes nous utilisons l'algorithme de tri par paquets (bucket sort) aussi dénommé tri par comptage, ou tri par base (radix sort). Ce sont des algorithmes qui peuvent utiliser au mieux la mémoire cache et qui peuvent être très facilement parrallélisables. Notre objectif n'est pas d'avoir la liste des suffixes triés mais d'avoir la liste des répétitions qui correspondent à des symboles.

E.2.1 Le tri par paquets

L'article de KÄRKKÄINEN et al. [KR09] « *Engineering Radix Sort for Strings* » décrit parfaitement ce type de tri. Le pseudo code de l'algorithme par paquets est donné page 59. C'est un algorithme qui s'appelle récursivement.

Notre choix s'est porté sur l'algorithme de tri par paquets car :
— comme le montre le pseudo code, **l'algorithme est très simple**,
— il est linéaire en temps, voir, l'article d'ANDERSSON et al. [AN94],
— il peut être facilement optimisé en tenant compte de l'utilisation des mémoires caches des ordinateurs récents, voir l'article de SEDGEWICK et al. [SW15],
— il est très fortement parallélisable (chaque paquet peut-être trié de façon indépendante), ce qui est un plus avec les nouveaux types d'ordinateurs.

E.2.2 Le lien entre les paquets et les répétitions

Tous les paquets de l'algorithme de tri, ont un préfixe commun aux suffixes du paquet, par exemple le paquet (2) (de l'exemple de la figure E.2) contient les 5 suffixes, « ISSISSIPPI$ », « ISSIPPI$ », « IPPI$ » et « I$ ». Tous ces suffixes ont le préfixe « I » en commun, de même, le paquet (2.3.1) contient les suffixes « ISSISSIPPI$ », « ISSIPP$ », tous ces suffixes ont en commun le préfixe « ISSI ».

Remarque : le préfixe d'un paquet est égal au préfixe du paquet situé à sa droite auquel on a rajouté la lettre qui a servi à extraire les suffixes. Par exemple le paquet (5.2) (de l'exemple de la figure E.2) de préfixe « SS » est égal au préfixe du paquet (5) de préfixe « S » auquel on a rajouté la lettre « S ».

LISTE DES SUFFIXES A TRIER		Tri sur la lettre numéro										SUFIXES TRIES	
		1		2		3		4		5			
		N° paquet	suffixes	N° paquet	suffixes	N° paquet	suffixes	N° paquet	suffixes	N° paquet	suffixes		
0	MISSISSIPPI$	1	**$**									11	$
1	ISSISSIPPI$	2	**ISSISSIPPI$**	2.1	**I$**							10	I$
2	SSISSIPPI$		**ISSIPPI$**	2.2	**IPPI$**							7	IPPI$
3	SISSIPPI$		**IPPI$**	2.3	**ISSISSIPPI$**	2.3.1	**ISSI**SSIPPI$	2.3.1.1	**ISSI**SSIPPI$	2.3.1.1.1	**ISSIP**PI$	4	ISSIPPI$
4	ISSIPPI$		**I$**		ISSIPPI$		**ISSI**PPI$		**ISSI**PPI$	2.3.1.1.2	**ISSIS**SIPPI$	1	ISSISSIPPI$
5	SSIPPI$	3	**M**ISSISSIPPI$									0	MISSISSIPPI$
6	SIPPI$	4	**PPI$**	4.1	**PI$**							9	PI$
7	IPPI$		**PI$**	4.2	**PPI$**							8	PPI$
8	PPI$	5	**S**SISSIPPI$	5.1	**S**ISSIPPI$	5.1.1	**SIP**PI$					6	SIPPI$
9	PI$		**S**ISSIPPI$		**SI**PPI$	5.1.2	**SIS**SIPPI$					3	SISSIPPI$
10	I$		**S**SIPPI$	5.2	**SS**ISSIPPI$	5.2.1	**SSI**SSIPPI$	5.2.1.1	**SSIP**PI$			5	SSIPPI$
11	$		**S**IPPI$		**SS**IPPI$		**SSI**PPI$	5.2.2.2	**SSIS**SIPPI$			2	SSISSIPPI$

FIGURE E.2 – Exemple de tri par paquets des suffixes du mot « MISSISSIPPI »

Avant de démontrer le lien biunivoque entre paquets et répétitions, précisons le lien qui existe entre un paquet et une répétition.

Nous avons vu que l'on associe un préfixe à chaque paquet. Si l'on prend le paquet (2) de préfixe « I » de l'exemple précédent, ce paquet contient les 5 suffixes qui commencent par le préfixe « I » (« ISSISSIPPI$ », « ISSIPPI$ », « IPPI$ », « I$ »), les positions de ces suffixes sont 1, 4, 7, 10. On peut dire que ce paquet correspond à la répétition $R_{S,u} = (p; i_1, i_2, ..., i_e)$ avec $U = I, p = 1, i_1 = 1, i_2 = 4, i_3 = 7, i_4 = 10$, ce qui nous donne $R_{S,I} = (p; 1, 4, 7, 10)$.

$R_{S,I}$ est la répétition associée au paquet.

Lemme 1 : les répétitions associées aux paquets tels que définies ci-dessus sont des répétitions complètes.

Démonstration : soit $R_{S,u} = (p; i_1, i_2, ..., i_e)$ une répétition associée à un paquet, si cette répétition n'est pas complète cela veut dire qu'il $\exists k$, tel que $S[k..k + p - 1] = u$, pour $k \in 1, 2, ..., n - p + 1$ et $k \ni i_1, i_2, ..., i_e$. Il existerait donc un préfixe u, d'un suffixe qui après le tri ne serait pas dans le même paquet que les autres suffixes commençant par ce préfixe ce qui est impossible. Le tri par paquets trie tous les suffixes de la chaîne S par ordre chronologique des lettres des suffixes, donc le préfixe u du suffixe $S[k..n]$ est obligatoirement dans le même paquet que tous les autres préfixes u des suffixes $R_{S,u} = (p; i_1, i_2, ..., i_e)$. Cette répétition associée au paquet est une répétition complète de S.

Lemme 2 : toutes les répétitions de S peuvent être reliées à un paquet.

Démonstration : soit $R_{S,u} = (p; i_1, i_2, ..., i_e)$ une répétition, cela veut dire que les suffixes ; $i_1, i_2, ..., i_e$ de S commencent par le préfixe u, et, dans le tri, ils seront rangés séquentiellement et ils seront contenus dans le paquet dont le préfixe est u.

Lemme 3 : tous les paquets sont associés à des répétitions de S et toutes les répétitions de S sont associées à des paquets.

Démonstration : ce lemme découle des 2 lemmes précédents.

Lemme 4 : les paquets les plus à droite avec au moins 2 suffixes représentent les plus longues répétitions NRE (No Right Extendible : non extensible à droite).

Démonstration : les paquets les plus à droite, sont les paquets dont le paquet précédent et le paquet suivant, quand ils existent, sont à gauche du paquet, et correspondent au niveau juste inférieur. Dans notre exemple « MISSISSIPPI » nous avons 2 paquets à droite avec au moins 2 suffixes : le premier paquet (2.3.1.1) de préfixe « ISSI » avec les suffixes « ISSIPPI » et « ISSISSIPPI », le deuxième paquet (5.2.1) de préfixe « SSI » avec les suffixes « SSIPPI » et « SSISSIPPI ».

Si la répétition du paquet le plus à droite était extensible à droite, cela voudrait dire qu'il existe un préfixe u' de deux des suffixes de S, tel que u soit le préfixe de u'. Si ces deux suffixes existaient, ils seraient obligatoirement dans

la liste des suffixes du paquet de préfixe u, et, comme ils sont 2 à avoir le préfixe u', ils donneraient naissance à l'itération suivante, à un paquet plus à droite avec 2 suffixes. Et par conséquent le paquet de préfixe u, considéré comme le paquet le plus à droite, ne serait pas le paquet le plus à droite.

La fréquence d'occurrence de répétition $R_{S,u} = (p; i_1, i_2, ..., i_e)$, est égale à e. C'est le nombre de fois où la sous-chaîne répétée est apparue. Par exemple, pour $R_{S,I} = (1; 10, 7, 4, 1)$, la fréquence d'occurrence de la sous-chaîne répétitive $u = I$ est égale à 4. La fréquence d'occurrence de la sous-chaîne s'écrit $freq(R_{S,u})$. Par construction, la fréquence d'occurrence de répétions associées à un paquet est égale au nombre de suffixes du paquet.

Identification des paquets contenant les symboles

En appliquant les lemmes précédents, pour localiser toutes les répétitions qui peuvent correspondre à des symboles, nous recherchons toutes les répétitions complètes ou non et surtout non extensibles à gauche (NLE). Il suffit de localiser les paquets les plus à droite de haute fréquence de répétition et qui répondent à un certain nombre de critères.

Nous cherchons donc les répétitions :
- non extensibles gauche (NLE)
- complètes ou non complètes
- dont les sous chaînes ne se chevauchent pas
- qui répondent aux critères fixés par l'utilisateur :
  - la période p (longueur des sous-chaînes) soit $\geq p_{min}$,
  - la fréquence $freq$ (nombre d'occurrences des sous-chaînes) soit $\geq freq_{min}$,
  - l'indice de confiance C soit $\geq C_{min}$.

La notion d'indice de confiance

Cette notion d'indice de confiance permet de savoir si le générateur u de la répétition a plus de probabilité d'être un symbole que de ne pas l'être . Pour cela, un indice de confiance C_σ est associé à chaque code σ de l'ensemble Σ, cet indice C_σ est tel que $0 \leq C_\sigma \leq 100$, il est initialisé à 50. Chaque fois qu'une répétition est validée par l'utilisateur comme étant un symbole, les indices de confiance des codes du générateur sont augmentés de 1, ou diminués de 1 dans le cas contraire, ceci dans les limites des bornes 0 et 100.

L'indice de confiance du générateur u, et donc de la répétition est $C_{(u)} = (\sum_{i=0}^{=p-1} c_i)/p$ avec $p =$ la période de u et où c_i est l'indice de confiance des codes de u.

Pourquoi définir des seuils minima

La méthode définie dans Péré-Laperne et al. [PC17] est incrémentale et itérative et repose sur la méthode KDD avec la participation active de l'utilisateur (les auteurs parlent d'Antropocentic Knowledge Discovey in Database). Le fait qu'elle soit itérative pour l'identification des symboles va nous permettre de mettre en place une statégie d'identification des symboles.

A la première itération l'utilisateur va rechercher les symboles qui ont le plus d'objets. Pour cela il prend $p_{min} \geq 30$, $freq_{min} \geq 20$ et $C_{min} \geq 50$, cette stratégie va permettre de trouver les symboles avec le plus d'objets (supérieurs à 30) et qui sont en grand nombre (supérieurs à 20). De cette façon, on est quasiment sûr que l'algorithme « SAP » va identifier tous les symboles qui correspondent à ces critères. La probabilité que les liaisons et les autres objets graphiques, qui sont dessinés de façon aléatoire, puissent répondre à ces critères, c'est à dire être présents plus de 20 fois avec exactement le même ordre pour 30 objets graphiques, est proche de 0.

Après identification, tous ces objets graphiques des répétitions sont exclus de la chaîne d'entrée et l'indice de confiance des codes est mis à jour. Les objets graphiques d'une répétition sont remplacés par le code de leurs symboles. Si le symbole est un symbole électrique, les segments de droite qui sont en liaison avec les points de connexion des équipotentielles sont considérés comme des liaisons (des équipotentielles) et ils sont remplacés par la notion de liaison.

Le fait de fixer $freq_{min} \geq 20$ permet de diminuer le temps du tri car les paquets de taille inférieure ne sont pas traités. KÄRKKÄINEN et al. [KR09] et SEDGEWICK et al. [SW15] considèrent qu'il faut passer au tri par insertion, pour des raisons de performance, quand la taille du panier est proche de $\sqrt{|\Sigma|}$ soit dans notre cas 16.

A l'itération suivante, l'utilisateur va modifier p_{min}, $freq_{min}$ et C_{min}. Il va obtenir de nouveaux résultats. Mais l'algorithme « SAP » est développé de telle façon que, si dans un paquet on rencontre un symbole ou une liaison, le paquet est immédiatement abandonné, car un symbole ne peut pas contenir un symbole ou une liaison. Ceci diminue très fortement les temps de calcul des itérations suivantes.

En faisant varier ces 3 paramètres avec des curseurs, l'utilisateur voit instantanément le résultat et peut ainsi valider l'identification des symboles qui sont rangés comme instance d'une ontologie.

Et ainsi de suite.

Modification du tri par paquets pour extraire les répétitions

Le tri basique par paquets (voir l'algorithme 1 « SAP » ci-dessus) subit les modifications suivantes :

- l'appel de la récursivité n'est exécuté que :
  - si le nombre de suffixes du paquet est $\geq freq_{min}$,
  - si la profondeur du paquet est $\leq L_{inf}$,
  - si l'objet à traiter est différent d'un symbole ou d'une liaison ($cOK = \text{true}$),
- lors de la construction du tableau des suffixes (SA) ou de l'arbre des suffixes, il faut traiter et trier tous les suffixes. Dans notre cas, on s'arrête bien avant, ce qui fait gagner un temps considérable,
- en fin de procédure récursive on n'enregistre une répétition que si :
  - la période de la répétition est $\geq p_{min}$,
  - le nombre d'objets dans le paquet est $\geq T_{inf}$,

— l'indicateur paquet le plus à droite est vrai,
— les occurrences ne se chevauchent pas,
— les occurrences sont NLE (non extensibles à gauche).