# Evaluation formative du savoir-faire des apprenants à l'aide d'algorithmes de classification : application à l'électronique numérique

Mariam Tanana

# Évaluation formative du savoir-faire des apprenants à l'aide d'algorithmes de classification

—

## Application à l'électronique numérique

**THÈSE**

Présentée et soutenue le 19 novembre 2009
pour l'obtention du

**Doctorat de l'INSA de Rouen**
**(Spécialité Informatique)**

par

**Mariam TANANA**

**Composition du jury**

| | | |
|---|---|---|
| Mustapha Bennouna | (président) | Président de l'Université Abdelmalak Essaâdi - Maroc |
| Nicolas Delestre | (encadrant) | Maître de Conférences à l'INSA de Rouen |
| Elisabeth Delozanne | (examinateur) | Maître de Conférences à l'Université Pierre et Marie Curie - Paris |
| Agathe Merceron | (rapporteur) | Professeur à Beuth University of Applied Sciences - Berlin |
| Jean-Pierre Pécuchet | (directeur) | Professeur à l'INSA de Rouen |
| Philippe Vidal | (rapporteur) | Professeur à l'Université Paul Sabatier - Toulouse |

*À Ahmed, Yassine et Laïla*

# Remerciements

Je tiens à remercier M. Mustapha Bennouna, président de l'Université Abdelmalak Essaâdi Tétouan-Tanger au Maroc, pour m'avoir permis de commencer cette thèse et aussi pour m'avoir fait l'honneur de la présider.

Je remercie Agathe Merceron, Professeur à Beuth University of Applied Sciences - Berlin et Philippe Vidal, Professeur à l'Université Paul Sabatier - Toulouse, d'avoir accepté de rapporter avec soin ce mémoire. Je remercie aussi Elisabeth Delozanne, Maître de Conférences à l'Université Pierre et Marie Curie - Paris, pour sa participation au jury, pour avoir manifesté un grand intérêt à mes travaux, et pour son soutien et sa sympathie.

Je remercie aussi Jean-Pierre Pécuchet, directeur de ma thèse, pour toute l'aide qu'il a pu me fournir tout au long de ces années. Je voudrais remercier tout particulièrement mon encadrant Nicolas Delestre qui m'a aidé à reprendre à temps les « rênes » de ma thèse et auprès duquel j'ai beaucoup appris. Je dédie aussi une pensée à Françoise Guégot, ancienne Maître de conférences associée à l'INSA de Rouen.

Je remercie aussi M. Abdellah Jabbouri, M. Saâd Cherif d'Ouazzane, anciens directeurs de l'ENSA Tanger et M. Abderrahmane Sbihi, directeur actuel de l'ENSA Tanger, pour leur soutien.

Je tiens à remercier l'ensemble du laboratoire LITIS pour m'avoir accueillie et permis de réaliser cette thèse, tout particulièrement : Brigitte Diarra, Florence Aubry et Sandra Le Bras pour m'avoir débarrassée de tous les soucis administratifs, Abdelaziz Bensrhair, Stéphane Canu, Nicolas Malandain, Sébatian Bonnegent, Jean-François Brulard pour m'avoir souvent bien accueillie, sans oublier tous ceux que j'ai rencontré pendant mes séjours au laboratoire.

Je remercie bien évidemment mon collègue Abderrahim Bourkane pour son aide pendant les séances d'expérimentations et pour son soutien incontestable. Il en va de même pour Abdelouahed Lyahyaoui et Réda Britel qui étaient toujours là quand j'en avais besoin.

Je remercie également mes autres collègues pour leur encouragement permanent, je cite tout particulièrement : Hafsa Benaboud, Saiida Lazaar, Rabia Gouy, Noureddine Motaki, Naoufal Sefiani, Amal Maurady et tous les autres.

Je voudrais aussi remercier certaines personnes que je n'ai connues que par e-mail interposé, mais qui ont répondu spontanément à toutes mes questions. Ils se reconnaîtrons

peut-être un jour, je cite : Randy Bryant, David Cox, Eric Salvat, Terry Scott, Christophe RAPINE, David Genest, Micehl Chein, Heather D. Pfeiffer, Carl Burch, Stéphane Garnaud, Serge Dusausay, Viviane Guéraud, Ruddy Lelouche, Olivier Hu, Mickael Rubens, Carlo Sansone, Donatello Conte, Patrick Maupin, Bertrand Cuissart, Frédéric Suard et tout spécialement Kaspar Riesen.

Je remercie aussi les anciens élèves-ingénieurs : Mustapha Eddine et Khalid Benomar pour leur participation au développement de la plate-forme TEB.

Finalement, je dédie ce travail à ma chère petite famille, mon mari Ahmed et mes enfants Yassine et Laïla, à mes parents Abdelkrim et Sakina, à mon oncle Mehdi et ma tante Kenza, à ma belle-mère Rabiâ et à la mémoire de mon beau-père, et à toute ma grande famille.

# Table des matières

# Table des figures

# Liste des tableaux

# Liste des algorithmes

# Introduction Générale

## Introduction

Grâce à l'évolution technologique, les Technologies de l'Information et de la Communication (TIC) ont pu intégrer divers domaines. L'éducation et la formation n'ont pas échappé non plus à cette avancée, avec l'émergence du nouveau secteur de l'enseignement/formation à distance. Ce secteur bénéficie, dans une large mesure, des progrès en cours, notamment l'utilisation de l'Internet comme moyen de communication à distance.

Depuis plusieurs années, les spécialistes de l'informatique et des sciences de l'éducation se sont beaucoup intéressés à ce nouveau secteur par l'élaboration d'outils et de logiciels pour l'apprentissage à distance. Toutefois, ils se sont davantage centrés sur la manière de **transmettre** les enseignements que sur leur **évaluation**, alors que celle-ci se situe au cœur même d'un enseignement.

En effet, au cours d'un processus d'apprentissage, les apprenants sont confrontés à plusieurs activités d'évaluation réalisées par leurs enseignants (Lieury, 1996). Ces évaluations peuvent être de natures différentes suivant le but recherché et apparaissent à différents moments d'un apprentissage afin de modifier les objectifs pédagogiques si nécessaire (Hadji, 1999). Un enseignant peut être amené à évaluer *avant*, *pendant* et *après* un processus d'apprentissage. Pour les enseignants, l'évaluation est un outil indispensable pour vérifier si les objectifs qu'ils ont définis ont été atteints par les apprenants.

Par ailleurs, l'introduction des TIC dans l'Éducation (TICE) vise deux publics et deux objectifs différents : l'apprenant, pour lequel on recherche à améliorer le processus d'apprentissage, et l'enseignant, dont on cherche à simplifier la tâche. Le développement des TICE ces dernières années a permis une évolution remarquable dans les pratiques de plusieurs aspects de l'enseignement à distance, notamment avec le développement des plates-formes. L'enseignant est amené à mettre à la disposition des apprenants des cours de différentes disciplines. Le nombre de cours à distance a augmenté d'une manière « spectaculaire ». Par contre, les concepteurs sont généralement plus soucieux de la qualité *technique* et *ergonomique* qu'ils offrent à leurs « clients » potentiels. Nous le voyons notamment dans les plates-formes d'enseignement à distance, où le souci majeur est le développement d'un outil informatique solide, portable et facile à utiliser et à administrer. Le respect d'une démarche pédagogique vient en deuxième lieu, voire entièrement ignorée, et uniquement si les contraintes techniques ont été résolues.

Pour ces mêmes raisons, le problème de l'évaluation des acquis des apprenants n'a pas encore été suffisamment abordé. Elle n'a toujours pas trouvé une place privilégiée dans de telles structures et n'est pas non plus une des priorités. La plupart du temps, les plates-formes par exemple, ne proposent que de simples outils de gestion de questionnaires.

Dans les domaines où un « savoir-faire » est nécessaire pour l'acquisition des connaissances, il est toujours difficile d'évaluer un apprenant. Nous pouvons prendre l'exemple des travaux pratiques où il est nécessaire que l'apprenant puisse manipuler pour pouvoir acquérir

un savoir-faire dans la discipline concernée. En dehors de la préparation des TP eux-mêmes, le travail le plus fastidieux pour l'enseignant reste l'évaluation des résultats fournis par les apprenants, surtout si l'effectif concerné est élevé.

### Contexte et problématique

Avant d'évaluer un apprenant, un enseignant doit définir le niveau cognitif des connaissances qu'il souhaite évaluer, par exemple : évaluer le « savoir »ou le « savoir-faire » d'un apprenant dans une discipline donnée. De cette manière, et en fonction de l'objectif pédagogique visé, il sera amené à utiliser des outils d'évaluation plus ou moins complexes.

La création d'un logiciel d'évaluation nécessite de définir les éléments suivants :

- le niveau de connaissance à évaluer ;
- les connaissances que l'on doit apporter au système à sa création ;
- les connaissances que l'on doit apporter au système à la création d'un exercice.

Par exemple, les QCM[1] (Labat, 2002) sont des systèmes simples, qui ne nécessitent pas beaucoup de connaissances à la création, mais demandent un investissement important à la création de chaque exercice.

D'un autre côté, lorsqu'un enseignant veut évaluer le « savoir-faire » des apprenants avec un outil informatique, il utilise souvent les Systèmes Tutoriels Intelligents (STI). Le développement des STI s'est fait en partie grâce aux avancées de l'Intelligence Artificielle dans le domaine de représentation des connaissances et de résolution des problèmes (Baron, 1994). Ils sont généralement basés sur les systèmes à base de règles (Bruillard, 1997; Buche, 2005), et donc très liés à un domaine d'application donné. De nombreux STI ont été développés, nous pouvons citer entre autres *Aplusix* (Nicaud, 1987; Nicaud *et al.*, 2007) pour l'apprentissage du calcul algébrique et *ANDES* (Conati *et al.*, 1997; Vanlehn *et al.*, 2005) pour la physique, etc. Par ailleurs, la création d'un système STI nécessite beaucoup de connaissances, mais très peu au moment de produire des exercices.

Malheureusement, et malgré le succès de plusieurs STI, il reste encore des limites à leur utilisation. En effet, les techniques des STI sont difficilement applicables à des domaines moins bien structurés que l'algèbre, la programmation, ou la géométrie par exemple, ou encore la représentation des connaissances par un expert dans un STI est généralement différente de celle des apprenants (Mendelsohn et Dillenbourg, 1993), etc . De plus, le fait de devoir expliciter les connaissances du domaine rend le développement des STI coûteux et difficile.

Notre objectif général est d'optimiser l'apport de connaissances à la création du système et à la création de chaque exercice. L'idéal serait de fournir peu de connaissances à la création du système et à celle des exercices.

[1] Questions à Choix Multiples.

## Objectif de recherche et contexte applicatif

Précédemment, F. Delorme(2005) a travaillé sur l'évaluation formative et sommative de l'apprenant en utilisant des algorithmes de classification par apprentissage supervisé. Le principe de ces algorithmes est d'essayer de déterminer l'étiquette d'une nouvelle donnée à partir d'un ensemble de données préalablement étiquetées (appelée « base d'apprentissage ») et d'une mesure de similarité entre ces données. Les systèmes utilisant de tels algorithmes ne demandent pas de connaissance *a priori* sur le domaine, puisque celle-ci est censée être contenue dans la base d'apprentissage. Ceci leur procure un avantage primordial sur d'autres méthodes où l'expertise doit être complètement explicitée. Par ailleurs, il est évident que plus le nombre de données d'apprentissage est important, meilleure est l'efficacité de ces algorithmes (en évitant toutefois de faire du « sur apprentissage »).

Dans ses travaux, F. Delorme s'est limité à l'évaluation formative et sommative du *savoir* des apprenants. Notre hypothèse est que ces mêmes algorithmes vont aussi nous permettre d'évaluer leur *savoir-faire*, afin de vérifier si les apprenants sont capable de mettre en pratique les connaissances qu'ils ont acquises.

C'est dans ce cadre que nous proposons un outil *d'aide à la correction* destiné à faciliter la tâche d'évaluation des apprenants à l'enseignant. L'outil pourra s'intégrer par la suite, comme une *composante* faisant partie d'un processus d'évaluation dans un EIAH (Environnement Informatique pour l'Apprentissage Humain) donné (Tchounikine, 2009).

Nous avons choisi comme domaine d'application l'électronique numérique où la mise en place des travaux pratiques est une phase obligatoire pour cet enseignement. Les apprenants sont amenés à travailler sur du matériel réel ou virtuel pour la réalisation et la conception de schémas électroniques. En dehors de la préparation des TP, le travail le plus fastidieux pour l'enseignant reste l'évaluation des résultats fournis par les élèves. En effet, ce n'est pas toujours évident de comprendre et corriger un schéma électronique réalisé par un apprenant, et encore moins, si le nombre des schémas est élevé.

## Organisation du mémoire

Ce mémoire est divisé en trois grandes parties. La première partie est constituée de trois chapitres sur l'état de l'art concernant les thèmes étroitement liés à notre thèse. La deuxième partie, constituée elle aussi de trois chapitres, présente de façon progressive les différentes expérimentations et résultats que nous avons obtenus pour valider notre hypothèse de travail. La troisième partie regroupe la conclusion, la bibliographie et toutes les annexes contenant la totalité des résultats expérimentaux.

Le **chapitre 1** dresse un état de l'art sur l'évaluation dans les processus d'apprentissage. Nous allons détailler dans ce chapitre les fonctions de l'évaluation et répondre à des questions qui sont nécessaires pour la planification et l'organisation d'un tel processus.

Le **chapitre 2** présente les méthodes de classification utilisées dans le domaine de l'apprentissage automatique (*Machine Learning*) et nous détaillerons quelques approches de classification supervisée, comme la méthode des k plus proches voisins, la classification bayésienne naïve, la méthode des Séparateurs à Vaste Marge (SVM) et la méthode basée sur les réseaux de neurones. Toutes ses méthodes seront illustrées par des exemples ou des algorithmes.

Le **chapitre 3** propose un état de l'art sur les outils existants dans le domaine de l'électronique numérique, notamment ceux qui permettent la création de schémas électroniques. Nous distinguerons ceux qui permettent d'effectuer des évaluations des apprenants par rapports à ceux qui ne le font pas. Ensuite, nous présenterons la plate-forme TEB (pour Travaux pratiques d'Électronique Binaire), que nous avons développée dans le but de gérer et d'évaluer les schémas électroniques des apprenants.

Le **chapitre 4** présente quelques définitions des circuits logiques, ainsi que des méthodes de représentation et de simplification des fonctions logiques. Nous définirons aussi dans ce chapitre la synthèse d'un circuit logique, illustrée par un exemple. Nous proposerons une représentation abstraite des schémas électroniques et nous finirons par présenter brièvement les circuits logiques manipulés et évalués dans la plate-forme TEB (Travaux pratiques d'Électronique Binaire).

Le **chapitre 5** commence par quelques définitions et notations générales sur les graphes. Ensuite, il présente quelques mesures de similarité existantes entre graphes. Pour finalement proposer une mesure de similarité entre schémas électroniques, validée à l'aide de données réelles.

Le **chapitre 6** présente les expérimentations utilisant la mesure de similarité entre schémas électroniques, un algorithme de classification choisi parmi ceux présentés dans le chapitre 2, et une base d'apprentissage plus complète. Dans un premier temps, cette base d'apprentissage est construite manuellement, ensuite, elle est générée automatiquement. Finalement, ces expérimentations sont étendues à d'autres exercices et nous terminerons par une interprétation des résultats obtenus.

Enfin, pour conclure ce mémoire, nous effectuerons un bilan de nos travaux et nous ouvrirons des perspectives de recherche.

# Première partie

# État de l'Art

# Chapitre 1

# L'Évaluation dans le processus d'apprentissage

**Sommaire**

## Résumé

L'évaluation prend une place très importante dans le processus d'apprentissage. Par contre, s'il fut un temps où une simple « note » suffisait pour évaluer un apprenant, aujourd'hui, l'évaluation est une tâche qui est planifiée à l'avance et fait partie intégrante du processus d'apprentissage.

En effet, au cours de leur processus d'apprentissage, les apprenants sont confrontés à plusieurs situations d'évaluation réalisées par leurs enseignants respectifs. Cette évaluation peut être de nature différente suivant le but recherché. Elle est en général effectuée sous le contrôle d'un enseignant qui devra ensuite avertir les apprenants des résultats obtenus.

L'évaluation, sous sa forme de pratique dominante, est toujours mal perçue par les apprenants. Ces derniers voient dans cet acte, comme le responsable d'un éventuel échec, ce qui la rend très impopulaire au sein de la communauté des apprenants. Pour les enseignants, l'évaluation est un outil indispensable pour vérifier si les objectifs qu'ils ont définis ont été atteints par les apprenants. Elle leur permet d'attribuer une « note » ou une « appréciation » au travail effectué par l'apprenant.

En résumé, l'évaluation est destinée aussi bien aux apprenants qu'aux enseignants, elle a pour objet de :

- guider l'apprenant dans son processus d'apprentissage ;
- donner des informations aux enseignants pour réajuster et améliorer leurs enseignements ;
- certifier les compétences acquises par l'apprenant ;
- détecter et corriger les éventuels dysfonctionnements d'un enseignement ;
- etc.

Dans la suite de ce chapitre, nous allons donc détailler les fonctions de l'évaluation et répondre à des questions qui sont nécessaires pour la planification et l'organisation de ce processus.

## 1.1 Petite histoire de l'évaluation

Les modes d'évaluation sont apparus en Europe vers le 16$^{e}$ siècle (Maulini, 1996). Initiés par les collèges des Jésuites, leur seul but était d'instaurer un système éducatif fortement sélectif (« *...il s'agit de privilégier les plus méritants et d'éliminer les autres...* »). Les meilleurs élèves, à la suite d'un classement élémentaire (par exemple, les copies corrigées étaient classées selon leur mérite), étaient récompensés par la « distribution d'un prix ».

Ensuite, l'utilisation *des appréciations chiffrées* a remplacé *les indications de rang*, tout en gardant la notion de classement, généralement en fin d'année scolaire (Maulini, 1996). À la suite, tout un arsenal de méthodes d'évaluation, par l'attribution de divers trophées, fait

son apparition pour stimuler de plus en plus la compétitivité et la rivalité entre les élèves. Ce n'est que vers la fin du 19$^{e}$ siècle, que sera officialisé par l'Etat, du moins en France, le système de notation de 0 à 20. Par ailleurs, face à l'augmentation croissante des effectifs des classes et d'autres raisons plus socio-économiques, l'école publique a été confrontée à divers changements tant au niveau des méthodes d'enseignement, qu'au niveau de l'évaluation. De nouveaux concepts apparaissent, comme la « moyenne » et le « redoublement » (Maulini, 1996).

Au début du 20$^{e}$ siècle, un nouveau mouvement de rationalisation de l'évaluation voit le jour (Barbier, 2001). L'intérêt au fonctionnement de l'acte d'évaluation contribue à l'apparition d'une nouvelle science : la « docimologie » ou « Science de l'évaluation ». La docimologie est constituée de deux approches différentes mais étroitement liées (Barbier, 2001) :

- la docimologie critique : qui s'occupe du fonctionnement des pratiques d'évaluation ;
- la docimologie prescriptive : qui s'occupe de l'amélioration du fonctionnement de ces pratiques.

L'école d'aujourd'hui se base encore sur les méthodes héritées des anciennes institutions, elle n'a pas encore trouvé une solution parfaite pour l'évaluation et la sélection scolaire. De maints chercheurs en font leur cheval de bataille et continuent à se poser les questions suivantes : faut-il supprimer définitivement le mode de notation actuel ? Et si oui, vers quel type d'évaluation faut-il s'orienter ? D'après J. Cardinet, un spécialiste de l'évaluation cité par Maulini (1996) : « *L'évaluation ne nécessite plus de classer les élèves, mais consiste simplement à savoir si chaque individu a atteint ou non l'objectif . . .* ».

## 1.2 Évaluation et pédagogie différenciée

Les techniques de différenciation pédagogique permettent à l'apprenant de suivre un parcours d'apprentissage adapté à ses besoins et à sa motivation, à la différence d'une pédagogie traditionnelle. Cette différenciation réside aussi bien dans l'apprentissage des notions que dans leur évaluation.

Ph. Perrenoud (2000) distingue dans son ouvrage deux méthodes d'enseignement :

- Une approche traditionnelle, qui est basée sur les méthodes traditionnelles d'enseignement, telles que nous les connaissons actuellement, en termes de cours magistraux, de devoirs et d'examens communs à tous les apprenants survenant à la fin d'un cycle d'apprentissage et notés par l'enseignant. À la suite de quoi, nous disposons d'un classement des apprenants : « des plus mauvais élèves » aux « meilleurs élèves » ;
- Une approche différenciée, basée sur l'individualisation des parcours de formation, centrée sur l'apprenant ou le modèle de l'apprenant, de telle façon que « *chaque apprenant se trouve, aussi souvent que possible, dans des situations d'apprentissage fécondes pour*

*lui, c'est-à-dire susceptibles de le faire progresser* ». Pour cela, il faut mettre en place des dispositifs de suivi individualisé des apprenants tels que : conseil, tutorat ou soutien.

Dans une pédagogie différenciée, l'évaluation doit aussi être conçue de façon individualisée. Celle-ci doit mener l'apprenant à valider ses divers acquis à travers son parcours pédagogique et doit tenir compte de tout état ou situation dans laquelle se trouve l'apprenant : « *c'est l'élève qui demande à être évalué, au moment de son choix et qui fixe le niveau de maîtrise qu'il prétend atteindre, . . . s'il réussit, il est certifié . . ., s'il échoue, cela n'a pas de conséquences graves, il continue à s'entraîner et se présente plus tard à une nouvelle évaluation* » (Perrenoud, 2000). L'objectif d'une telle évaluation n'étant plus de « juger » ou de « sanctionner » l'apprenant, mais plutôt de l'aider à « progresser » dans son apprentissage. Même le droit à l'erreur est reconnu dans une pédagogie différenciée : « *On est globalement passé d'une conception négative donnant lieu à sanction, à une autre où les erreurs se présentent plutôt comme indices pour comprendre le processus d'apprentissage et comme témoins pour repérer les difficultés des élèves* » (Astolfi, 2001). D'autres pédagogues ont aussi réhabilité le rôle de « l'erreur » dans une pédagogie différenciée (Jorro, 2000) :

- *L'erreur est consubstantielle à l'acte de connaître* ;
- *L'erreur est un trop plein de connaissances* ;
- *L'erreur informe* ;
- *L'erreur dépend de la logique du sujet* ;
- *L'erreur est positive, elle est acte* ;
- *L'erreur est discriminée par la classe* ;
- *L'erreur est analysée* ;
- *L'erreur appelle une réflexion du praticien sur sa démarche* ;
- *L'erreur est liée au statut de son producteur* ;
- *L'erreur appelle une approche différenciée.*

D'une façon générale, avant d'évaluer un apprenant, un enseignant doit se poser certaines questions (Hadji, 2000). Les plus importantes, à notre niveau, seront explicitées dans les sections qui vont suivre.

## 1.3 Pourquoi faire une évaluation ?

L'évaluation est une affaire de « communication » qui met généralement en relation trois grands acteurs : l'enseignant, l'apprenant et l'institution (Jorro, 2000). L'enseignant est chargé par l'institution de transmettre les savoirs, l'apprenant est censé acquérir ces savoirs, et l'institution est responsable d'informer les parents (ou la société en général) du « degré d'appropriation » de ses savoirs par les apprenants. L'évaluation est décrite par J.-M. Barbier (2001) comme un « *un acte délibéré et socialement organisé aboutissant à la production de jugements de valeur* ».

J. Cardinet (1988), en réponse à cette question, distingue quatre buts essentiels :

- *améliorer les décisions relatives à l'apprentissage de chacun des élèves* ;
- *informer l'élève et ses parents sur sa progression* ;
- *décerner les certificats nécessaires* ;
- *améliorer l'enseignement en général*.

Cette vision traditionnelle de l'évaluation est toujours d'actualité, même si les nouvelles pédagogies tendent, tant bien que mal, à la rendre plus au service des apprenants (Perrenoud, 2000).

Cette évaluation peut être **implicite** ou **explicite**. D'après A. Lieury (1996), les apprenants sont très souvent évalués pendant leur processus d'apprentissage que ce soit d'une façon implicite, sous format de remarques, gestes (mimiques) ou appréciations employées par l'enseignant envers le comportement et le travail des apprenants, ou d'une façon explicite, lorsqu'il s'agit d'attribuer une note à un travail donné. Il définit l'évaluation comme « *un jugement de valeur énoncé à partir d'informations recueillies par un observateur* ».

Finalement, si nous voulons synthétiser les réponses possibles pour cette question, nous pouvons attribuer deux grandes missions pour l'acte d'évaluation :

- une mission sociale, qui vise la certification, l'orientation, ou la sélection des apprenants ;
- une mission pédagogique, qui vise le diagnostic, l'interprétation, et la remédiation des difficultés d'apprentissage des apprenants.

## 1.4 Pour évaluer quel niveau de connaissance ?

En (1956), Bloom distingue trois grands domaines d'apprentissage : cognitif, affectif et psychomoteur. Le domaine cognitif décrit la connaissance et le développement des habilités et compétences intellectuelles (le savoir et le savoir-faire), le domaine affectif décrit les aptitudes ou le savoir-être, et le domaine psychomoteur décrit les habilités physiques et motrices. Dans ce qui suit, nous allons nous intéresser au domaine cognitif puisqu'il est le plus sollicité au moment de la mise en place des objectifs d'apprentissage dans le système éducatif.

Bloom a proposé une taxonomie (classification hiérarchisée) qui permet de classer les objectifs pédagogiques (ou d'apprentissage) en 6 niveaux. Tous ces objectifs se traduisent en comportements attendus des apprenants et ces niveaux sont classés en fonction de la complexité cognitive de l'objectif visé. Nous distinguons donc :

1. **La connaissance** (le niveau le plus inférieur) où l'apprenant est capable de prendre connaissance d'une information et de la restituer quand c'est nécessaire (par exemple : connaître et se rappeler des concepts, des faits, des définitions, des procédures, etc.) ;

2. **La compréhension** où l'apprenant est capable de traiter, comprendre ou interpréter une information en fonction de ce qu'il a appris ;
3. **L'application** où l'apprenant est capable d'utiliser une aptitude ou une connaissance, acquise préalablement, dans une nouvelle situation (par exemple : appliquer une procédure, une règle, résoudre un problème mathématique, construire un graphe ou une carte, etc.) ;
4. **L'analyse** où l'apprenant est capable d'analyser et raisonner dans une situation donnée (par exemple : présenter un rapport à partir d'une recherche, intervenir dans une discussion professionnelle, etc.) ;
5. **La synthèse** où l'apprenant est capable de regrouper des connaissances issues de plusieurs domaines pour créer une production personnelle. La synthèse relève de la créativité (par exemple : réaliser une carte conceptuelle, création artistique ou intellectuelle, etc.) ;
6. **L'évaluation** (le niveau le plus haut) où l'apprenant est capable d'évaluer et de juger les processus qui l'ont conduit à faire, en fonction de normes et de critères précis.

Le niveau « connaissance » étant la base pour les cinq autres. Sans savoir (connaissance), il ne peut y avoir aucune compréhension. Sans compréhension, il ne peut y avoir aucune application. Sans compréhension ni application, il ne peut y avoir aucune analyse ou synthèse. Et sans aucune expérience en analyse et synthèse (notamment dans la résolution de problème), il ne peut y avoir aucune évaluation, ni comparaison avec d'autres solutions.

Il existe néanmoins d'autres taxonomies qui répartissent les objectifs pédagogiques en plusieurs niveaux. Nous pouvons citer entre autres :

- Lorin W. Anderson et David R. Krathwohl proposent une taxonomie révisée de Bloom (Anderson *et al.*, 2001) ;
- La taxonomie de Van Hiele[1], spécialisée dans l'apprentissage de la géométrie ;
- La taxonomie à 3 niveaux de De Landsheere (1984) ;
- et celles de R. Mills Gagné (1965), Ph. Meirieu (2002), etc.

Même si la taxonomie de Bloom est assez générale (Landsheere, 1984), parce que *transdisciplinaire*, elle demeure de loin la plus utilisée. C'est pour cela que nous allons l'adopter.

Finalement, lorsqu'un enseignant veut évaluer un apprenant, il détermine d'abord l'objectif pédagogique visé, ce qui l'amène à utiliser différents outils d'évaluation. Par exemple, pour la taxonomie de Bloom, nous avons la correspondance « objectifs pédagogiques-outils d'évaluation » dans la figure 1.1.

[1]Source : http ://fr.wikipedia.org/wiki/Taxonomie_de_van_Hiele

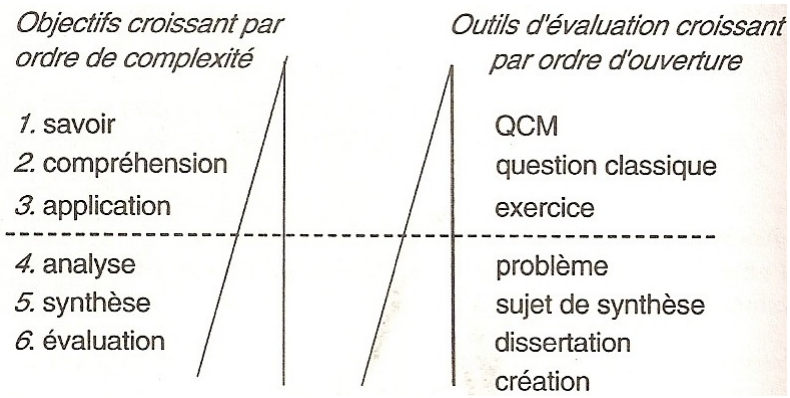



FIG. 1.1 – Un exemple de correspondance : objectifs pédagogiques-outils d'évaluation (Abernot, 2002)

## 1.5 Quand faire une évaluation ?

Dans une démarche d'enseignement différencié, une évaluation des acquis est nécessaire à différents moments d'un apprentissage afin de modifier les objectifs pédagogiques si nécessaire. Un enseignant peut donc évaluer **avant**, **pendant** et **après** un processus d'apprentissage. Ces différentes évaluations auront donc des objectifs différents suivant le moment de leur déroulement. Dans l'ensemble, il existe trois grandes fonctions, ou trois grands « moments » d'évaluation qui seront détaillés dans les paragraphes suivants.

### 1.5.1 Évaluation pronostique

Le but de ce type d'évaluation est de vérifier si l'apprenant a les connaissances nécessaires pour poursuivre une formation donnée. Pour cela, l'évaluation pronostique est généralement effectuée au début ou en fin de formation pour des formations ultérieures (Jorro, 2000; Hadji, 2000). Elle est aussi source d'informations pour l'apprenant afin de l'orienter vers un type de formation donnée où il aurait plus de chance de réussir.

Ce type d'évaluation peut être utilisé dans un processus d'apprentissage pour définir le niveau de l'apprenant afin de construire son profil et de l'orienter vers les informations qui lui font défaut.

### 1.5.2 Évaluation sommative

C'est l'évaluation qui aboutit à la certification d'un module, d'une unité de valeur ou à l'obtention d'un certificat à l'issue d'une période de formation (Jorro, 2000; Hadji, 2000). Elle est réalisée lorsqu'un enseignant a besoin d'attribuer des notes aux apprenants afin de *« certifier qu'ils ont atteint un certain niveau de formation »* (Lieury, 1996).

En général, elle se situe à la fin d'une période de formation et son but est de montrer si l'apprenant a été capable d'atteindre les objectifs visés par la formation (*« l'ensemble des sa-*

*voirs : savoirs théoriques, méthodologiques et pratiques* » (Lieury, 1996)). Elle peut aussi apparaître tout au long du processus de formation : « *Une simple note, si elle prétend repérer un niveau assez général d'acquisitions, relève de l'évaluation sommative* » (Lieury, 1996). Si après une évaluation sommative, il y a élaboration d'un diplôme ou d'un certificat particulier, à la fin d'un cycle d'études par exemple, nous aboutissons à une certification sociale (Lieury, 1996).

### 1.5.3 Évaluation formative

Le terme d'évaluation formative a été proposé par M. *Scriven* (1967). C'est une évaluation qui se produit à n'importe quel moment au cours d'un processus d'apprentissage. Elle est à visée essentiellement pédagogique. Son but est de guider et de réguler l'apprenant dans son apprentissage, ainsi que de positionner les connaissances acquises (ou performances accomplies) par l'apprenant par rapport à un objectif d'apprentissage donné (Perrenoud, 1999; Jorro, 2000; Hadji, 1999, 2000).

L'intérêt de ce type d'évaluation réside dans le diagnostic des difficultés qui peuvent subvenir à l'apprenant durant son processus d'apprentissage. Ce qui permet à l'enseignant d'analyser et d'interpréter les résultats afin de trouver les causes probables de ces difficultés, et d'adapter ou réguler le parcours de l'apprenant en conséquence, en insistant sur les parties qui permettront de surmonter les difficultés rencontrées. C'est la phase de « remédiation ».

Comme l'a défini A. Lieury (1996) : *L'évaluation formative peut donc être caractérisée par un cycle « constat de difficultés-interprétation-remédiation »*. D'une certaine manière, l'évaluation formative réhabilite l'erreur, car celle-ci devient un facteur déterminant dans le processus d'apprentissage, ne dit-on pas « qu'on apprend de ses propres erreurs ».

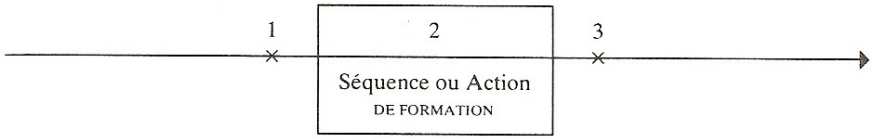
La différence entre les évaluations sommative et pronostique par rapport à l'évaluation formative est que cette dernière ne vise pas à donner une note à l'apprenant. Son but est d'établir des observations qualitatives qui doivent être suffisamment fiables afin d'y remédier le mieux possible (Lieury, 1996).

### 1.5.4 Chronologie « évaluative »

Finalement, l'acte d'évaluation s'insère dans un processus d'apprentissage en tenant compte du facteur chronologique, car il peut avoir différents sens, selon où il se place dans ce processus (Hadji, 2000) (cf. figure 1.2).

## 1.6 Comment faire une évaluation ?

Pour les trois types d'évaluation définis ci-dessus, nous pouvons soit définir avec précision les objectifs de la formation (évaluation critériée), soit s'intéresser au classement des apprenants les uns par rapports aux autres (évaluation normative) (Lieury, 1996; Hadji, 1999).




| En (1) : AVANT L'ACTION DE FORMATION | En (2) : PENDANT L'ACTION | En (3) : APRÈS L'ACTION DE FORMATION |
|---|---|---|
| **Évaluation :**<br>● Diagnostique<br>● Pronostique<br>● Prédictive<br>**Fonction :**<br>● Orienter<br>● Adapter | **Évaluation :**<br>● Formative<br>● « Progressive »<br>**Fonction :**<br>● Réguler<br>● Faciliter (l'apprentissage) | **Évaluation :**<br>● Sommative<br>● Terminale<br>**Fonction :**<br>● Vérifier<br>● Certifier |

FIG. 1.2 – Fonctions de l'évaluation selon sa place par rapport à l'action de formation (Hadji, 2000)

Très souvent, il est aussi question de définir des critères d'évaluation. Ces notions seront explicitées dans les paragraphes suivants.

### 1.6.1 Évaluation critériée

L'évaluation critériée n'attribue pas de note à l'apprenant, elle se base sur une grille d'évaluation pour savoir si l'apprenant a atteint les objectifs pour une situation d'apprentissage donnée. Pour cela, il faut définir au préalable les différents critères « évaluables » qui feront partie de la grille d'évaluation.

### 1.6.2 Évaluation normative

L'évaluation normative consiste à évaluer un apprenant au sein d'un groupe. Les apprenants sont ensuite classés les uns par rapport aux autres suivant une norme établie au départ. L'évaluation est donc entièrement basée sur la performance d'un apprenant par rapport à une collectivité d'apprenants intervenant sur le même sujet. Par exemple, le système de notation ECTS[2] qui a pour objectif d'unifier la notation pour des programmes d'études entre différents pays européens.

Ce système de points a été développé par l'Union Européenne et a pour but de faciliter la lecture et la comparaison des programmes d'études des différents pays européens. Une échelle de notation classe les performances des apprenants sur une base statistique (cf. http ://fr.wikipedia.org/wiki/ECTS).

[2]European Credits Transfer System.

### 1.6.3 Critères d'évaluation

Le petit Larousse illustré définit le terme « évaluation » dans l'enseignement, comme : « *Mesure à l'aide de critères déterminés des acquis d'un apprenant.* ». En effet, pour avoir une évaluation homogène, il faudra disposer d'une grille (appelée « grille d'évaluation ») où seront mentionnés tous les critères qui seront évalués dans une situation donnée.

Afin de construire une grille d'évaluation, il est nécessaire d'établir auparavant un référentiel de ce qu'on attend de l'apprenant. D'après C. Hadji (1999) : « *L'évaluation exige la construction de ce qui a été désigné comme son référent, à savoir un ensemble de critères spécifiant un système d'attentes. Chaque critère définit ce qu'on estime être légitimement en droit d'attendre de l'objet évalué* ».

Pour De Peretti (2000), l'évaluation ne concerne pas uniquement la vérification de l'acquisition d'un savoir, elle peut servir aussi à observer les comportements, la qualité d'une méthode de travail, etc. Pour cela, il distingue plusieurs indicateurs ou critères qui pourront faire partie d'une grille d'évaluation. Le résultat qui découle de la grille d'évaluation servira à indiquer si l'apprenant a atteint l'objectif d'apprentissage visé.

L'établissement d'une grille d'évaluation est difficile et délicat car elle dépend de nombreux facteurs. Les critères doivent être fixés par les enseignants à partir de réalisations observables et en fonction des objectifs d'apprentissage visés. De Peretti (2000) distingue les types de critères suivants : les critères quantitatifs et les critères qualitatifs.

#### 1.6.3.1 Critères quantitatifs :

Ces critères peuvent être liés à des valeurs mesurables, comme par exemple :

1. Le temps de réponse mis pour répondre à chaque question ;
2. Le temps global pour répondre à l'ensemble des questions, en incluant les temps de réflexion intermédiaires ;
3. Évaluation terminée dans les délais ;
4. Les résultats à des questions fermées ;
5. Les résultats à des questions semi-ouvertes ;
6. etc.

#### 1.6.3.2 Critères qualitatifs :

Ces critères sont plutôt liés à des valeurs observables, difficilement mesurables, comme par exemple :

1. Temps de résolution limité ;
2. Employer des sources d'informations ;

3. Évaluation individuelle ou en groupe ;
4. La démarche stratégique de résolution utilisée ;
5. Comportement face à une situation imprévue ;
6. etc.

Finalement, il n'existe pas de grille passe-partout. L'enseignant construit sa propre grille et définit des critères particuliers en fonction des objectifs d'un processus d'apprentissage.

## 1.7 Avec quels outils faire une évaluation ?

Nous disposons d'un grand choix de types de question que nous pouvons utiliser dans les évaluations. Nous pouvons distinguer : les « questions ouvertes » où l'apprenant répond avec ses propres mots et les « questions fermées » où la (ou les) réponse(s) se trouve(nt) parmi une liste de choix proposée à l'apprenant.

Les questions fermées sont plus faciles à implémenter et sont corrigées d'une manière objective, elles sont surtout utilisées dans des situations où l'on veut tester les connaissances de l'apprenant sur un sujet particulier. Par contre, comme elles sont créés avec des « mots » de l'enseignant, il peut y avoir des problèmes d'interprétation de la part de l'apprenant.

Les questions ouvertes servent plus à mettre l'apprenant dans une situation où il pourra s'exprimer avec son propre style, faire preuve de prise de décision personnelle, de créativité et de communication. Elles sont beaucoup plus subjectives et donc plus difficiles à corriger.

Par la suite, nous présentons une description des différents types de question utilisés dans les évaluations.

### 1.7.1 Questions fermées

L'inconvénient de ce type de questions est qu'elles ne permettent à l'apprenant, ni de s'exprimer ni de rédiger avec son propre texte. Nous distinguons :

#### 1.7.1.1 Les QCM (Question à Choix Multiples)

Un QCM est composé d'un énoncé et de plusieurs réponses possibles. C'est la forme d'exercices la plus simple. L'apprenant répond en choisissant (cochant) une ou plusieurs réponses. Il existe plusieurs variantes de QCM : vrai/faux, trouver la seule bonne réponse, trouver la meilleure réponse, trouver la seule réponse fausse, trouver toutes les bonnes réponses, etc (Lepage et Romainville, 2009). Leur correction est simple et rapide, elle peut être effectuée de façon automatique.

Les QCM ont été intensivement utilisés dans les évaluations pédagogiques car ils sont faciles à implémenter et permettent un traitement rapide des réponses. Par ailleurs, il leur est

souvent reproché d'être surtout adapté à l'évaluation des connaissances, d'inciter au « bachotage » et de ne pas solliciter les capacités d'expression des apprenants (Lepage et Romainville, 2009; Lieury, 1996).

#### 1.7.1.2 Question de type association ou appariement

Ces questions proposent à l'apprenant deux listes de choix où il devra faire correspondre ou associer des éléments de la première liste avec un ou plusieurs éléments de la deuxième liste.

Nous pouvons avoir des associations graphiques à réaliser, comme par exemple déplacer un élément de la première liste dans un emplacement qui correspondrait à un élément de la deuxième liste, ou relier par un trait graphique un objet de la première liste avec un autre objet de la deuxième liste, etc. Nous pouvons aussi avoir des associations textuelles, les deux listes étant référencées par des indices alphanumériques et il suffit d'avoir une troisième liste avec les associations possibles.

La complexité de la question est augmentée si l'appariement entre les deux listes n'est pas bijective, c'est-à-dire, si un élément de la première liste peut être associée à plusieurs éléments de la seconde, et réciproquement.

#### 1.7.1.3 Question de type réarrangement

Ces questions proposent à l'apprenant de mettre dans un certain ordre une liste d'éléments qui font partie de l'énoncé. Celui-ci contient des propositions dans un ordre quelconque. L'apprenant doit alors les « réarranger » dans le bon ordre.

### 1.7.2 Questions semi-ouvertes

Ce type de questions est un compromis entre le type précédent et le type suivant, il ne permet pas à l'apprenant de s'exprimer en son propre langage, mais, celui-ci a la possibilité de répondre de plusieurs façons différentes. La réponse à ce genre de question doit être brève et précise, et pour laquelle on donne des orientations. Nous distinguons :

#### 1.7.2.1 Question à réponse courte

L'apprenant répond à la question par un ou plusieurs mots. L'enseignant pose des questions claires et précises qui imposent des réponses brèves et spécifiques. Généralement, les réponses correspondent à des mots ou de courtes expressions.

#### 1.7.2.2 Question de type « carte conceptuelle »

Les cartes conceptuelles sont des graphes qui permettent de représenter les concepts d'un domaine de connaissances donné et les liens entre ces concepts. Elles ont été introduites à l'origine par Novak et Gowin (1984).

Ce formalisme a été utilisé par F. Delorme (2005) pour le développement d'un outil générique pour l'évaluation des connaissances de l'apprenant pour les hypermédias adaptatifs. Cet outil permet de générer un rapport complet sur la production de l'apprenant, permettant ainsi de savoir si l'apprenant a compris ou non une notion du cours et éventuellement les difficultés rencontrées afin d'y remédier.

F. Delorme définit 2 types de construction des cartes conceptuelles par les apprenants :
- La construction complète d'une carte conceptuelle : l'apprenant devra choisir les concepts et les liens qui doivent y figurer, ainsi que la structure de la carte ;
- La construction partielle d'une carte conceptuelle, dont la structure, les concepts et les liens ont été fixés. L'apprenant doit simplement placer les concepts et/ou liens au bon endroit sur la carte.

L'enseignant commence par modéliser les notions qu'il souhaite aborder à l'aide d'une carte conceptuelle. La carte de l'enseignant est utilisée comme référence pour évaluer les cartes des apprenants. L'évaluation est basée sur les différences existant entre les deux cartes.

D'autres travaux de recherche ont aussi utilisé les cartes conceptuelles pour représenter les conceptions d'un apprenant, nous pouvons citer entre autres (Tribollet *et al.*, 2000; Pudelko *et al.*, 2003).

### 1.7.3 Questions ouvertes

Les questions ouvertes sont plus compliquées à mettre en œuvre et à corriger automatiquement, sachant que c'est l'apprenant qui répond librement à la question :
- en tenant compte des données du problème à résoudre ;
- en respectant la méthode de résolution ;
- en interagissant avec le système pour modifier les paramètres du problème ;
- etc.

Le principal avantage de ce type de question est la possibilité d'évaluer des objectifs pédagogiques de niveau cognitif élevé (cf. Taxonomie de Bloom). Il invite les apprenants à s'exprimer avec leur propre langage et leur style. Par contre, la correction de telles questions est subjective et plus longue à réaliser. Pour le bon déroulement de ce type d'évaluation, il faut respecter certaines règles ou procédures pour l'élaboration de telles questions. L'énoncé doit être clair et précis, et contenir toutes les informations pour la résolution du problème.

### 1.7.4 Conclusion

Il existe plusieurs systèmes qui ont intégrés ces outils d'évaluation. Certains systèmes utilisent un seul type d'outil, par exemple des QCM (Labat, 2002), d'autres peuvent utiliser plusieurs types différents. Par exemple, PÉPITE (Jean-Daubias, 2000) est un système qui propose l'évaluation des connaissances et compétences des élèves en fin de collège dans le domaine des mathématiques (l'algèbre plus précisement). Il propose une grande variété d'exercices, des plus simples (QCM vrai/faux), aux plus complexes : des questions ouvertes, où l'apprenant doit justifier en même temps sa réponse. Il dispose en outre d'un outil auteur supplémentaire (Prévit, 2008) qui lui permet de générer de façon automatique ou assistée des banques d'exercices de diagnostic pour plusieurs niveaux de compétence, avec leurs solutions anticipées sous forme de grilles d'analyse. Ces solutions peuvent être correctes ou erronées. Les résultats des élèves sont associés à une des solutions anticipées pour ainsi obtenir une caractérisation de ces résultats.

## 1.8 Les incertitudes de l'évaluation

Ce sont les études docimologiques qui s'occupent des problèmes d'évaluation et de leur caractère « subjectif » dû aux divergences existantes entre les différents enseignants au moment des corrections. En effet, pour établir un jugement évaluatif, les informations qui sont collectées doivent être fiables et pertinentes. Plusieurs expériences ont montré qu'il y a diverses causes possibles de cette subjectivité. Ces causes sont responsables de l'absence de fiabilité des procédures d'évaluation. C. Hadji (2000) en a recensé quelques unes :

- différents enseignants ne corrigent pas les mêmes productions de la même façon ;
- les évaluations attribuées à une production peuvent varier dans le temps avec le même enseignant (on parle alors d'infidélité de l'enseignant) ;
- les évaluations peuvent varier selon l'humeur, la disponibilité ou l'état de fatigue de l'enseignant ;
- le jugement de l'enseignant peut être influencé, inconsciemment, par des signes sociaux ou comportementaux des apprenants (on parle alors de favoritisme ou son contraire) ;
- l'évaluation d'un ensemble de productions peut être influencée par l'ordre dans lequel elles sont évaluées (on parle de l'effet d'ancre ou de stéréotypie ) ;
- etc.

Plusieurs méthodes ont été mises en place au cours des années pour améliorer l'objectivité ou la fiabilité des évaluations, par exemple : la multi-correction, la concertation entre les enseignants afin d'établir un barème à utiliser par tous, l'utilisation de procédures dites de

« modération »[3] qui consiste à modifier les notes attribuées afin de supprimer les différences de sévérité (moyenne) ou les différences dans l'utilisation de l'échelle des notes (variance), etc. En effet, la pertinence d'une évaluation sera d'autant meilleure que les objectifs poursuivis par l'enseignant seront définis avec précision. Dans ce cas, ces objectifs peuvent devenir les critères de l'évaluation, on parle alors d'évaluation critériée (Lieury, 1996).

A. Lieury (1996) distingue deux aspects de cette pertinence : le premier concerne les informations fournies par l'évaluation et qui doivent être assez pertinentes pour nous renseigner sur les difficultés rencontrées par les apprenants. Le second aspect est relatif à l'intérêt pédagogique de ces informations qui doivent nous permettre de mettre en place une procédure de remédiation efficace et adaptée à chaque apprenant puisque les apprenants ne rencontrent pas obligatoirement les mêmes difficultés au même instant (Lieury, 1996).

## 1.9 L'évaluation : Normes et standards

Comme nous l'avons vu, un acte d'évaluation fait partie intégrante d'une action de formation globale (professionnelle ou institutionnelle). La démarche qualité concernant l'évaluation est donc obligatoirement incluse dans celle de l'action de formation. De Peretti(2000) parle « d'ingénierie d'évaluation » qui doit être aussi différenciée et adaptée à chaque situation :« *la notion d'ingénierie vise, formellement, la mise en concordance, en optimisation des conceptions ministérielles, des procédures institutionnalisées, des recherches scientifiques, des attentes sociales, avec les pratiques personnelles de divers acteurs d'un système (dans notre cas, en matière d'évaluation). il convient d'en étudier les principales caractéristiques et les divers ingrédients ou composants, dans un champ de possibilités explorables, pour permettre des choix et des adaptations ou des innovations*. »

Depuis 1997, plusieurs groupes de chercheurs ont travaillé sur les bases de la standardisation concernant le domaine de l'apprentissage (Rodríguez-Estévez *et al.*, 2003). Nous pouvons citer entre autres, le Learning Technologies Standardization Comittee[4] de l'IEEE, l'IMS[5] Global Learning Consortium (*Instructional Management System*), l'AICC[6] (*Aviation Industry CBT Committee*) et le groupe ADL[7] (*Advanced Distributed Learning*), du département de la défense américain, qui est à l'origine du standard SCORM[8] (*Sharable Content Object Reference Model*).

Les spécifications à l'origine de ces travaux ont pour finalité la normalisation des méthodologies, services et caractéristiques dans le domaine de l'apprentissage. De telle manière, le déroulement d'une action de formation, et *a priori* des activités pédagogiques qui la consti-

[3] La modération ne modifie pas l'ordre des élèves établi par le correcteur, elle modifie seulement la moyenne et la dispersion de leurs notes.
[4] http ://ltsc.ieee.org
[5] http ://www.imsglobal.org
[6] http ://ltsc.ieee.org
[7] http ://www.adlnet.org
[8] Modèle de référence pour des objets de contenu partageables.

tuent (pour nous, c'est l'action d'évaluation qui nous intéresse), s'effectue de manière identique quelle que soit la plate-forme d'exécution.

## 1.10 Conclusion

Comme nous l'avons cité plus haut, lorsqu'un enseignant veut évaluer un apprenant, il commence par déterminer l'objectif pédagogique visé. En suite, en fonction du niveau cognitif de cet objectif, il va choisir un ou plusieurs types d'exercice. Plus le niveau cognitif à évaluer est « haut », plus il faut laisser de liberté à l'apprenant.

Nous souhaitons évaluer le *savoir-faire* de l'apprenant, qui correspond au niveau cognitif *application* de la taxonomie de Bloom. Nous allons donc nous intéresser à des outils adaptés à ce choix.

# Chapitre 2

# Les méthodes de classification

**Sommaire**

## Résumé

Dans ce chapitre, après une introduction aux méthodes de classification utilisées dans le domaine d'apprentissage automatique[1] (*Machine Learning* en anglais), nous présenterons quelques approches de classification supervisée.

Nous commencerons par la méthode des $k$ plus proches voisins, suivie de l'approche de classification bayésienne naïve, qui est un cas particulier de la modélisation à l'aide de réseaux bayésiens. Nous continuerons par la méthode des Séparateurs à Vaste Marge (SVM) et nous finirons par la méthode basée sur les réseaux de neurones. Toutes ces méthodes seront illustrées par des exemples ou des algorithmes.


## 2.1 Quelques définitions

Avant de présenter les méthodes de classification et leurs champs d'application, nous allons introduire quelques notions et définitions qui seront utilisées dans ce chapitre. Pour cela, nous allons reprendre quelques définitions qui sont disponibles dans (Preux, 2008) :

Soit :

- $\mathcal{X}$ : un ensemble de données ou d'objets et $N = |\mathcal{X}|$ ;
- $\mathcal{D}$ : espace contenant toutes les données. Nous avons la relation $\mathcal{X} \subset \mathcal{D}$ ;
- $\mathcal{A}$ : ensemble des attributs caractérisant chacune des données et $P = |\mathcal{A}|$ ;
- $\mathcal{E}$ : ensemble des étiquettes. $\forall e \in \mathcal{E}$, $e$ est appelé une « classe » ou « étiquette ».

**Définition 1** : Soit un ensemble $\mathcal{X}$ de $N$ données étiquetées. Chaque donnée $x_i$ est caractérisée par $P$ attributs et par sa classe $e_i \in \mathcal{E}$. Dans un problème de classification, la classe prend sa valeur parmi un ensemble fini. Le problème consiste alors, en s'appuyant sur l'ensemble d'exemples $\mathcal{X} = \{(x_i, e_i)_{i \in \{1,\dots,N\}}\}$, à prédire la classe de toute nouvelle donnée $x \in \mathcal{D}$.

**Définition 2** : Un « exemple » est une donnée pour laquelle on dispose de sa classe.

**Définition 3** : Une « classe » regroupe un ensemble de données ayant des attributs ou caractéristiques semblables.

**Définition 4** : Un « classifieur » est un algorithme qui, à partir d'un ensemble d'exemples, prédit la classe de toute nouvelle donnée.

**Définition 5** : Une « base d'apprentissage » est un ensemble de classes. Chaque classe est composée de données préalablement étiquetées de façon identique.

[1] http ://fr.wikipedia.org/ : « *L'apprentissage automatique est un des champs d'étude de l'intelligence artificielle. Il fait référence au développement, à l'analyse et à l'implémentation de méthodes qui permettent à une machine (au sens large) d'évoluer grâce à un processus d'apprentissage, et ainsi de remplir des tâches qu'il est difficile ou impossible de remplir par des moyens algorithmiques plus classiques* ».

## 2.2 Les méthodes de classification

Les méthodes de classification sont des outils indispensables et nécessaires à la communauté scientifique qui opère dans le champ d'application de l'**apprentissage automatique** (*Machine Learning* en anglais) (Mitchell, 1997). L'objectif général de ces méthodes est de regrouper les données étudiées en plusieurs classes différentes. Ensuite, à chaque nouvelle donnée, ces méthodes cherchent à prédire la classe d'appartenance de cette donnée parmi les classes déjà disponibles.

Les domaines d'application sont nombreux et variés, nous pouvons citer entre autres, la reconnaissance des formes (*Pattern Recognition*), la fouille ou l'entrepôt de données (*Data Mining, Datawarehouse*), la catégorisation de documents textuels, la recherche d'information sur le Web, la classification d'objets, la prise de décision automatisée (diagnostic médical, gestion des prêts bancaires), etc.

Il existe deux grandes approches pour les méthodes de classification : l'approche ou classification « supervisée » et l'approche ou classification « non-supervisée ». Dans les deux cas, nous avons besoin généralement d'une mesure ou d'une fonctionnalité entre données pour prédire leur appartenance à une classe particulière.

Ces deux approches comprennent une phase d'apprentissage ou de classification[2] : cette phase correspond à la création de classes à partir d'exemples ayant des attributs connus. L'ensemble de ces classes constitue la base d'apprentissage.

La classification supervisée dispose en plus d'une phase de décision ou de généralisation. Durant cette phase, en présence d'une nouvelle donnée, le classifieur devra prédire l'appartenance de cette donnée à une classe de la base d'apprentissage.

### 2.2.1 Classification supervisée

Dans la classification supervisée, les classes sont connues *a priori*. Chaque classe dispose d'une sémantique bien particulière qui la différencie des autres classes et regroupe un ensemble de données étiquetées avec les attributs de la classe.

Il existe de nombreuses techniques de classification supervisée, nous pouvons citer entre autres :

- les $k$ plus proches voisins ;
- les réseaux bayésiens naïves ;
- les réseaux de neurones ;
- les Séparateurs à Vaste Marge (*Support Vector Machine* en anglais) ;
- les arbres de décisions ;

[2] Le terme de classification est aussi bien utilisé pour l'action de classer (diviser et répartir en classes) que pour l'action de classifier (répartir selon une classification).

– la programmation génétique.

Cette liste n'est pas exhaustive. Dans les sections suivantes, nous allons détailler certaines de ces méthodes.

### 2.2.2 Classification non-supervisée

Dans la classification non-supervisée, aussi appelée segmentation (*clustering* en anglais), les classes ne sont pas connues *a priori*. Elles sont construites à partir de certaines règles ou critères de regroupement qui dépendent des données disponibles à un moment donné. Les classes sont généralement fondées sur la structure des données, la sémantique associée à chaque classe est donc plus difficile à déterminer.

Il existe aussi de nombreuses techniques pour la classification non-supervisée, nous pouvons citer entre autres :

– la méthode des K-moyennes (*K-means* en anglais) (MacQueen, 1967) : soit un entier $K$, cette méthode partitionne les données en $K$ classes disjointes. Elle commence par initialiser $K$ exemples dans l'espace de données de départ, également appelés « barycentres ». Ensuite, chaque donnée est affectée à la classe du barycentre le plus proche. La méthode calcule alors pour chaque classe le nouveau barycentre. Ces deux dernières étapes sont répétées de manière itérative, jusqu'à ce que le système se retrouve dans une position stable, c'est-à-dire, sur deux itérations successives tous les exemples restent dans la même classe ;

– Les cartes de Kohonen (aussi appelée carte auto-organisatrice ou *Self Organizing Map* en anglais) (Kohonen, 1982) sont des réseaux de neurones à une seule couche. Chaque neurone possède un élément de l'ensemble $\mathcal{D}$ nommé « prototype ». Les cartes sont utilisées pour obtenir des mécanismes de réduction de dimension de l'espace des données et aussi pour faire de la classification non-supervisée. Dans ce dernier cas, le nombre de neurones de l'unique couche est égale au nombre de classes (à chaque classe est associé bijectivement un neurone).

  Lorsqu'une donnée $x$ est présentée à une carte de Kohonen, un neurone est sélectionné. C'est celui dont le prototype est le plus proche de $x$, il est appelé le « neurone gagnant ». Dans la phase d'utilisation de la carte, la classe de $x$ est celle du neurone gagnant.

  La phase d'apprentissage d'une carte de Kohonen est assez simple. Initialement, les prototypes de chaque neurone sont initialisés aléatoirement. Chaque donnée de la base d'apprentissage est alors présentée aléatoirement plusieurs fois à la carte de Kohonen. Les prototypes des neurones au voisinage du neurone gagnant (ce dernier inclu) sont alors « rapprochés » de la donnée présentée.

### 2.2.3 Choix du type de classification

Il n'existe pas de méthode supérieure à toutes les autres. L'objectif est soit d'apprendre par l'exemple, et donc nous parlerons de classification supervisée ; soit d'extraire de l'information, et dans ce cas nous parlerons de classification non-supervisée. Dans tous les cas, le choix d'une méthode se fera en fonction du problème que nous voulons résoudre.

Notre objectif étant d'effectuer l'évaluation de schémas électroniques produits par les apprenants, en les comparant à des schémas électroniques de référence fournis par l'enseignant. Nous nous trouvons donc devant un cas d'apprentissage supervisé, puisque les classes sont connues à l'avance.

## 2.3 Les méthodes de classification supervisée

Dans les sections suivantes, nous allons présenter quelques méthodes de classification supervisée qui sont parmi les plus utilisées dans l'apprentissage automatique.

### 2.3.1 Les $k$ plus proches voisins

#### 2.3.1.1 Principe

La méthode des $k$ plus proches voisins ($k$ppv en abrégé, *k nearest neighbor* en anglais) est l'une des méthodes les plus simples de la classification supervisée. Elle fait partie de la catégorie des méthodes à noyaux (Duda *et al.*, 2001). Ces méthodes regroupent un ensemble d'algorithmes pour l'analyse statistique de données. Elles se caractérisent par l'utilisation d'une fonction (appelée « noyau ») permettant de mesurer la similarité entre les objets à analyser.

Le $k$ppv prédit la classe d'appartenance d'une donnée en recherchant dans la base d'apprentissage un ou plusieurs cas similaires. C'est la base d'apprentissage constituée des exemples déjà connus, associée à une fonction de distance, qui constitue le classifieur. À la différence des autres méthodes de classification qui seront étudiées dans les sections suivantes, il n'y a pas de phase d'apprentissage.

#### 2.3.1.2 Description

Le fonctionnement du classifieur est le suivant : étant donnée une base d'apprentissage correctement initialisée et un entier $k$, le $k$ppv détermine la classe d'un nouvel objet en lui attribuant la classe **majoritaire** des $k$ plus proches exemples dans la base d'apprentissage :

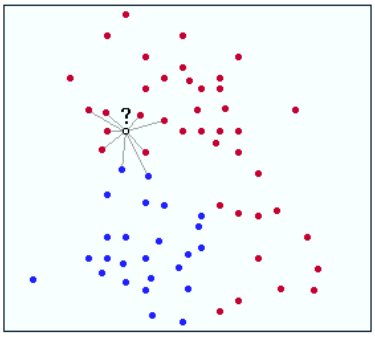



FIG. 2.1 – Recherche des $k$ plus proches voisins

Pour exprimer cette notion de proximité entre les données de la base d'apprentissage, une distance est associée au $k$ppv. D'un point de vu mathématiques, une distance est une application définie par :

$$\begin{aligned} \delta : \mathcal{D} \times \mathcal{D} &\rightarrow \mathbb{R}^+ \\ (x, y) &\rightarrow \delta(x, y) \end{aligned}$$

qui respecte les 3 propriétés suivantes $\forall x, y, z \in \mathcal{D}$ :

- $\delta(x, y) = 0 \iff x = y$
- $\delta(x, y) = \delta(y, x)$
- $\delta(x, y) + \delta(y, z) \geq \delta(x, z)$

Dans le cas des attributs numériques, un exemple de distance couramment utilisée est la distance euclidienne :

$$\forall x, y \in \mathbb{R}^n \quad \delta(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

#### 2.3.1.3 Algorithme du $k$ppv

Cela se traduit par l'algorithme suivant (Preux, 2008) :

**Algorithme 1** Les $k$ plus proches voisins

**Données:**

$x \in \mathcal{D}$, avec $\mathcal{D}$ un ensemble de données ;
$\mathcal{X}$ un ensemble de données étiquetées ;
$k \in \{1, ..., N\}$ ;
$\delta$ une mesure de similarité définie sur $\mathcal{D}$.

**Résultats:**

La classe de $x$.

1: **pour chaque** $x_i \in \mathcal{X}$ **faire**
2: calculer $\delta(x_i, x)$
3: **fin pour**

4: **pour chaque** $\kappa \in \{1, ..., k\}$ **faire**
5: kppv[$\kappa$]$\leftarrow argmin_{x_i \in \mathcal{X}}\ \delta(x_i, x)$
6: $\delta(x_i, x) \leftarrow +\infty$
7: **fin pour**

8: **retourner** La classe de $x$ à partir de la classe majoritaire des exemples qui sont stockés dans le tableau kppv.

La valeur de $k$ est un paramètre important lors de l'utilisation de l'algorithme du $k$pvv. Par contre, ce dernier n'est pas obligatoirement plus performant lorsque $k$ est grand (Duda *et al.*, 2001). Pour pouvoir prendre une décision de classification, il faut absolument que $k$ soit égale à $n \times |\mathcal{E}| + 1 \ \ avec, \ n \in \mathbb{N}$ *et* $\mathcal{E}$ ensemble des classes ou étiquettes. Pour simplifier, nous prenons $n = 0$ et donc $k = 1$, c'est-à-dire, le cas où le classifieur retournera la classe du plus proche voisin. L'algorithme correspondant est :

**Algorithme 2** Le 1 plus proche voisin

**Données:**

$x \in \mathcal{D}$, avec $\mathcal{D}$ un ensemble de données ;
$\mathcal{X}$ un ensemble de données étiquetées ;
$\delta$ une mesure de similarité définie sur $\mathcal{D}$.

**Résultats:**

La classe de $x$.

1: **pour chaque** $x_i \in \mathcal{X}$ **faire**
2: calculer $\delta(x_i, x)$
3: **fin pour**
4: $x_{1-ppv} \leftarrow argmin_{x_i \in \mathcal{X}}\ \delta(x_i, x)$
5: **retourner** La classe de $x_{1-ppv}$

#### 2.3.1.4 Avantages et inconvénients

Un des avantages majeurs de cette méthode est la simplicité de sa mise en œuvre. En effet, la phase d'apprentissage est inexistante, il suffit de stocker un ensemble d'exemples, préalablement étiquetés par un expert, dans la base d'apprentissage pour pouvoir démarrer la phase de décision. Par la suite, l'introduction de nouveaux exemples permet d'améliorer la qualité de cette méthode sans le besoin de reconstituer toute la base d'apprentissage.

Le *k*ppv est une méthode qui détermine la « proximité » d'une donnée vis à vis de la base d'apprentissage, ce qui lui procure un avantage certain au moment de l'interprétation des résultats de classification. D'un autre côté, si la distance obtenue est trop grande, le résultat de la classification peut être rejeté.

Par contre, la phase de décision est relativement lente en terme de temps de calcul, surtout si le nombre d'exemples de la base d'apprentissage est important. En effet, il est nécessaire de calculer la distance des données à classifier par rapport à tous les exemples de la base d'apprentissage. La complexité de l'algorithme est en $\mathcal{O}(N)$, auquel il faut multiplier la complexité de calcul de la distance. Par contre, les performances de l'implémentation sont améliorées en utilisant des techniques de structuration de la base d'apprentissage, permettant de réduire le nombre d'exemples classés qui sont à comparer avec les données.

### 2.3.2 Les réseaux bayésiens naïfs

#### 2.3.2.1 Principe

Les réseaux bayésiens sont des modèles de représentation des connaissances, utilisés pour l'aide à la décision, le diagnostic ou le contrôle de systèmes complexes (Naïm *et al.*, 2004). Les méthodes de classifications basées sur ces réseaux reposent sur une approche probabiliste utilisant la règle de Bayes $P(A|B) = \frac{P(A,B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$. La représentation graphique des réseaux bayésiens est fondée sur le formalisme des « graphes causaux ». Ce sont des graphes orientés acycliques, auxquels sont associées des probabilités conditionnelles liées aux arcs, et dont les nœuds représentent divers évènements pouvant s'influencer mutuellement (Naïm *et al.*, 2004).

La figure 2.2 présente un exemple quelconque de réseaux bayésiens. La survenue de l'évènement $J$ par exemple, dépendra de tous les évènements qui le précèdent dans le graphe, en fonction des probabilités de leurs survenues respectives.

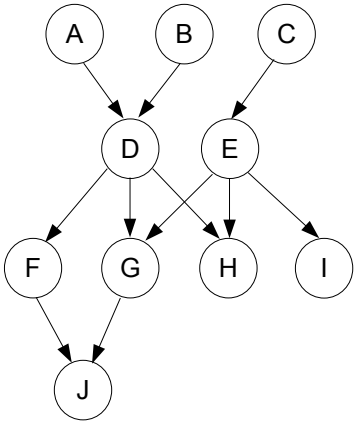



FIG. 2.2 – Exemple d'un réseau bayésien quelconque

#### 2.3.2.2 Description et algorithmes

La règle de Bayes stipule que, si l'évènement B peut être influencé par l'évènement $A$, alors le fait que $B$ a eu lieu permet également de prévoir la survenue de $A$.


FIG. 2.3 – Cas où $B$ peut être influencé par $A$

Avec, $P(A)$ et $P(B)$ les probabilités pour que les évènements $A$ et $B$ se produisent respectivement. $P(A, B)$ représente la probabilité pour que $A$ et $B$ surviennent tous les deux. $P(A|B)$ (respectivement $P(B|A)$) est la probabilité de survenue de l'évènement $A$, sachant que l'évènement $B$ a eu lieu (respectivement la probabilité de survenue de l'évènement $B$, sachant que l'évènement $A$ a eu lieu).

Par exemple, prenons le cas où $B$ est l'évènement « j'ai de la température », et $A$ l'évènement « j'ai la varicelle ». si $P(B|A) = 0,9$, nous dirons que la probabilité que « j'ai de la température sachant que j'ai la varicelle » est de 90%.

Un cas particulier des réseaux bayésiens est la modélisation bayésienne naïve, où nous disposons de $n$ évènements observables indépendants les uns des autres[3], c'est-à-dire, sans influence entre eux, et d'un évènement dont nous souhaitons calculer la probabilité de survenue. Nous obtiendrons une structure telle que celle de la figure 2.4.

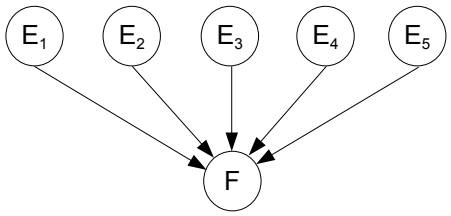



FIG. 2.4 – Exemple de réseau bayésien naïf

Cette approche naïve de la classification a été particulièrement développée dans les applications de classification de texte et notamment dans la détection des messages non désirables : « spam ». Pendant la phase d'apprentissage, l'expert propose au classifieur une série de textes qui correspondent à des spam et à des non-spam, afin que ce dernier apprenne à les distinguer. De cette manière, le classifieur analyse chaque texte pour en extraire les mots les plus fréquemment utilisés dans les deux catégories et attribue à ces mots les probabilités d'occurrences dans chaque cas. L'objectif à la fin de cette phase étant de connaître pour chaque mot, les probabilités qu'il apparaisse dans les messages spam et les messages non-pam.

L'algorithme correspondant à cette phase pour l'exemple des messages spam est :

[3] d'où le terme « naïf ».

**Algorithme 3** Phase d'apprentissage bayésienne naïve : spam et non-spam

**Données:**

Liste de messages des catégories spam et non-spam.

**Résultats:**

Liste de mots avec leurs probabilités respectives pour les catégories spam et non-spam.

1: **pour chaque** message de Liste de messages **faire**
2: **pour chaque** mot de message **faire**
3: **si** message est un spam **alors**
4: **calculer** $p(mot = rencontré|catégorie = spam)$
5: **sinon**
6: **calculer** $p(mot = rencontré|catégorie = non - spam)$
7: **fin si**
8: **fin pour**
9: **fin pour**

Pendant la phase de décision, le classifieur est capable d'identifier, avec plus ou moins de succès, si un message correspond à un spam ou non. Donc, pour chaque message $M$, il détermine les probabilités $p(catégorie = spam|message = M)$ et $p(catégorie = non - spam|message = M)$. Sachant que le message $M$ peut être décomposé en $n$ mots $m_i$ indépendants ($1 \leq i \leq n$), nous avons :

$$p(catégorie = spam|message = M) = \prod_{i=1}^{n} p(catégorie = spam|m_i = rencontré)$$

Ensuite, grâce à la règle de Bayes et à la phase d'apprentissage, il est possible de connaître la valeur de chaque terme de ce produit :

$$p(catégorie = spam|m_i = rencontré) = \frac{p(categorie=spam,\, m_i=rencontre)}{p(m_i=rencontre)}$$
$$= \frac{p(m_i=rencontre|categorie=spam)p(categorie=spam)}{p(m_i=rencontre)}$$

Finalement, le message $M$ sera considéré comme un spam si :

$$p(catégorie = spam|message = M) > p(catégorie = non - spam|message = M),$$

et comme non-spam dans le cas contraire.

L'algorithme correspondant est :

**Algorithme 4** Phase de classification bayésienne naïve : détection de spam

**Données:**

$m$ : le message à vérifier.

**Résultats:**

La catégorie du message, spam ou non-spam.

1: $p_{spam} \leftarrow p(catégorie = spam | message = m)$
2: $p_{non-spam} \leftarrow p(catégorie = non - spam | message = m)$
3: **si** $p_{non-spam} \geq p_{spam}$ **alors**
4: **retourner** catégorie non-spam
5: **sinon**
6: **retourner** catégorie spam
7: **fin si**

#### 2.3.2.3 Avantages et inconvénients

Cette méthode de classification est qualifiée d'incrémentale car à chaque nouvel exemple ajouté dans la base d'apprentissage, nous pouvons raffiner les probabilités (Preux, 2008). Elle a besoin d'une phase d'apprentissage pour le calcul des différentes probabilités *a priori* et d'une phase de décision qui correspond au classement d'un nouvel exemple. Cette dernière phase a une complexité en $\mathcal{O}(1)$, c'est la phase d'apprentissage qui est plus longue car elle nécessite d'effectuer les différents calculs de probabilité sur tous les exemples présentés par l'expert.

D'un autre côté, l'échec du classifieur survient généralement s'il est confronté à un trop grand nombre de mots qu'il n'a jamais rencontré pendant la phase d'apprentissage.

### 2.3.3 Les Séparateurs à Vaste Marge (SVM)

#### 2.3.3.1 Principe

Les Séparateurs à Vaste Marge (*Support Vector Machines* en anglais) ont été développés à partir des considérations théoriques introduites par V. Vapnik (1998). Ce sont des techniques d'apprentissage destinées à résoudre des problèmes de classification supervisée binaire, c'est-à-dire, que la base d'apprentissage est constituée uniquement de deux classes (0 et 1, +1 et -1, ou rouge et bleu, etc.). Ces techniques consistent à séparer les deux classes de la base d'apprentissage par un séparateur optimal, une droite linéaire appelée *hyperplan*.

#### 2.3.3.2 Description

Le classifieur SVM nécessite une phase d'apprentissage qui consiste à trouver l'Hyperplan Séparateur Optimal qui sépare les deux catégories de classes (figure : 2.5). Chaque exemple étant alors représenté par un vecteur de descripteurs de dimension $n$. Pendant cette phase, le classifieur recherche l'hyperplan H qui sépare les deux catégories de données de manière à ce que la distance entre les deux hyperplans H+ et H- (voir figure : 2.6) soit la plus grande possible. L'écart entre les deux hyperplans H+ et H- est appelé la « marge » (Preux, 2008). Cette marge est calculée de telle façon que la distance entre le point rouge le plus proche de H- et le point bleu le plus proche de H+, soit maximale.

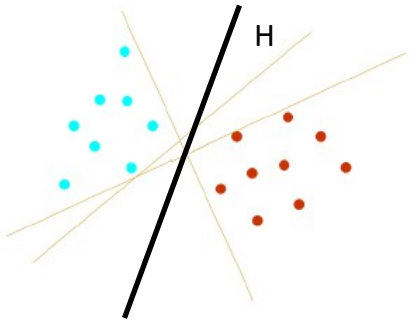



FIG. 2.5 – Exemple d'un SVM linéaire

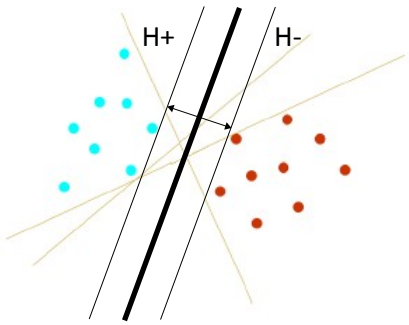



FIG. 2.6 – Exemple de marge d'un SVM linéaire

À la phase de classification, les nouveaux exemples peuvent se situer d'un côté ou l'autre de l'hyperplan, sans obligatoirement être trop similaires à ceux utilisés pour trouver l'hyperplan.

#### 2.3.3.3 Avantages et inconvénients

Les Systèmes à Vastes Marges sont des techniques de classification qui ont de très bonnes performances dans de nombreux domaines. La complexité de ces algorithmes est, dans le pire des cas, en $\mathcal{O}(N^3)$ pour la phase d'apprentissage et en $\mathcal{O}(N)$ pour la phase de décision.

Le principal inconvénient des SVM réside dans le choix de la fonction noyau adaptée au contexte en cours. De plus, le classifieur a été conçu à l'origine pour des problèmes à 2 classes. Les extensions actuelles aux problèmes mutli-classes sont souvent construites sur la base « d'une classe contre toutes les autres ». La durée de la phase d'apprentissage dans ce cas est très longue. D'un autre côté, l'interprétation des résultats pour un problème particulier n'est pas toujours facile. En effet, le principe de classification des SVM réside dans le fait qu'un nouvel exemple se trouve d'un côté ou de l'autre de la « marge », même s'il est techniquement loin des exemples étiquetés de sa classe d'appartenance.

### 2.3.4 Les réseaux de neurones

#### 2.3.4.1 Définitions et principe

Avant de décrire les méthodes de classification à base des réseaux de neurones, nous allons commencer par quelques définitions qui vont être utilisées dans le reste de cette section et qui ont été recueillies de (Denis et Gilleron, 2000; Dreyfus *et al.*, 2004) :

**Définition 1 :** Un neurone est une fonction non linéaire, paramétrée, à valeurs bornées.

**Définition 2 :** Un neurone élémentaire peut manipuler des valeurs binaires ou réelles. Différentes fonctions peuvent être utilisées pour le calcul de la sortie. Le résultat de la sortie peut être déterministe ou probabiliste.

Un neurone est représenté graphiquement par la figure 2.7[4]. Le neurone calcule la somme de ses entrées $x_i$, pondérées respectivement par $w_{ij}$, appelé « poids synaptique ». La valeur de cette somme passe à travers la fonction d'activation pour produire la sortie du neurone. Si le résultat est supérieur à un certain *seuil*, la sortie est alors « activée », généralement sa valeur vaut 1, dans le cas contraire, la sortie n'est pas activée, sa valeur vaut alors 0.

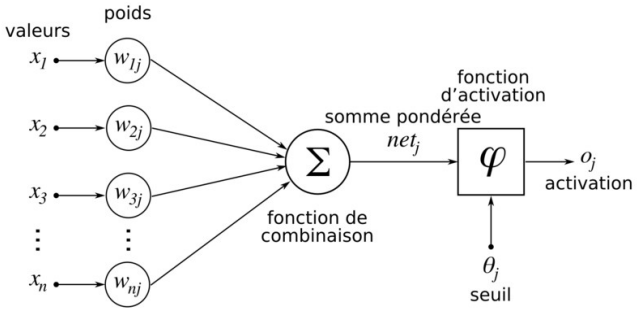



FIG. 2.7 – Structure graphique d'un neurone

**Définition 3 :** Un réseau de neurones est constitué d'un grand nombre de neurones de base interconnectés.

[4] Source : http ://fr.wikipedia.org/

**Définition 4 :** L'architecture du réseau peut être sans rétroaction, c'est-à-dire, sans boucle. La sortie d'un neurone ne peut alors influencer son entrée. Elle peut aussi être avec rétroaction totale ou partielle.

**Définition 5 :** Un réseau de neurones non bouclé réalise une ou plusieurs fonctions de ses entrées, par composition des fonctions réalisées par chacun des neurones.

Un réseau de neurones est représenté graphiquement par la figure 2.8[5]. Chaque neurone est interconnecté avec d'autres neurones pour former des « couches ». La couche d'entrée recueille les données à l'entrée du réseau. La couche de sortie présente les résultats calculés par le réseau. Des couches intermédiaires, aussi appelées « couches cachées », propagent le traitement des données entre les neurones jusqu'à la couche de sortie.

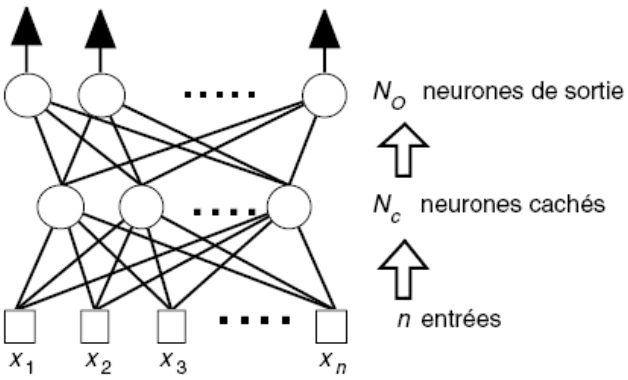



FIG. 2.8 – Structure graphique d'un neurone

Les $x_i$ représentent les attributs des données à classifier. Les neurones de sortie représentent les classes. La classe d'une donnée est celle du neurone le plus activé. Il y a autant d'entrées que d'attributs sur les données, et autant de neurones de sortie qu'il y a de classes.

**Définition 6 :** Le fonctionnement du réseau peut être synchrone : tous les neurones calculent leurs sorties respectives simultanément. Le fonctionnement peut être asynchrone : les neurones calculent leurs sorties chacun à son tour en séquence ou de façon aléatoire.

#### 2.3.4.2 Description

Les méthodes de classification à base de réseaux de neurones sont des techniques de calcul dont la modélisation est fortement influencée par le fonctionnement des neurones biologiques. Elles se sont inspirées au départ des travaux de Fr. ROSENBLATT (1958) sur l'organisation et le stockage des informations dans le cerveau humain. ROSENBLATT a développé en 1958 le modèle de « perceptron ». Ce modèle possède deux couches de neurones : une couche d'entrée qui sert à recevoir les données à traiter et une couche de sortie donnant le résultat

[5] Source : (Dreyfus *et al.*, 2004)

du problème à résoudre. Par contre, le perceptron est adapté uniquement à des données qui sont linéairement séparables (Denis et Gilleron, 2000).

Ensuite, des années plus tard, David E. Rumelhart et James L. McClelland introduisirent le « perceptron multi-couches » pour contrer la limitation du perceptron en traitant aussi les données non-linéairement séparables (Rumelhart et McClelland, 1986). Cette fois-ci le réseau de neurones est constitué d'un ensemble de couches de neurones fonctionnant en parallèle. Les données proposées au réseau, au niveau de la couche d'entrée, se propagent grâce aux neurones à travers les couches intermédiaires jusqu'à la couche de sortie. Les calculs sont effectués des entrées vers la ou les sorties. La propagation se fait de façon parallèle, dans la mesure où il n'y a pas de connexions entre les neurones de la même couche. En effet, seuls les neurones appartenant à des couches successives sont interconnectés.

La phase d'apprentissage de cette méthode consiste à choisir la structure du réseau et de ses paramètres, c'est-à-dire, le nombre de couches cachées et le nombre de neurones dans chacune de ces couches. Une fois ces choix fixés, le réseau peut être *calibré* avec des données proposées par l'expert. Au cours de cette opération, les poids des connexions du réseau et le seuil de chaque neurone seront modifiés au fur et à mesure pour pouvoir s'adapter à différentes conditions d'entrée. Une fois la phase d'apprentissage terminée, on passe en phase de décision pour réaliser le travail pour lequel il a été conçu.

Plusieurs algorithmes des méthodes à base de perceptron ou à base de perceptron multi-couches sont proposées dans (Gilleron et Tommasi, 2000; Denis et Gilleron, 2000; Preux, 2008).

#### 2.3.4.3 Avantages et inconvénients

La méthode de classification basée sur les réseaux de neurones est difficile à utiliser dans la pratique, elle nécessite beaucoup d'expérience et de connaissances techniques dans le domaine. C'est aussi une méthode qui a besoin d'un temps d'apprentissage qui peut être long, par contre, la phase de décision est plus rapide. Pendant la phase d'apprentissage, le calibrage du réseau, c'est-à-dire, le choix de la structure et des paramètres du réseaux, le nombre de couches, le nombre de neurones, etc., n'est pas évident, et on aboutit généralement à un choix optimal par essais successifs (Preux, 2008).

Un autre inconvénient majeur de cette méthode, c'est la difficulté d'interprétation du résultat de la classification par l'utilisateur. Il est très difficile d'extraire un modèle de l'apprentissage effectué par le réseau, c'est pour cela que l'on parle parfois de « boîte noire » (Preux, 2008).

## 2.4 Conclusion

Dans ce chapitre, nous avons présenté quelques méthodes de classification supervisée avec leurs principaux avantages et inconvénients, et ce afin de choisir une méthode adaptée

à notre contexte actuel. Par la suite, nous définissons les critères suivants pour choisir la méthode de classification que nous utiliserons dans nos travaux :

- facilité d'interprétation des résultats de classification ;
- simplicité d'utilisation et d'implémentation ;
- mise à disposition par l'expert d'un ensemble d'exemples de référence *a priori*, sans besoin d'une phase d'apprentissage.

Finalement, aucune de ces méthodes de classification n'est supérieure à une autre. Suivant le besoin et le domaine d'application, nous serons tenté d'utiliser une méthode plutôt qu'une autre. Par contre, il y en a une qui permet de faire du « rejet par distance », c'est le k-ppv. C'est une méthode simple, ne nécessitant pas d'apprentissage préalable et grâce à la notion de proximité, elle permet d'effectuer une interprétation des résultats de la classification, tout en rejetant certaines données si la distance obtenue par rapport aux données d'apprentissage est trop grande.

# Chapitre 3

# Les logiciels d'électronique numérique

**Sommaire**

## Résumé

Dans ce chapitre, nous allons dresser un état de l'art sur les outils existants dans le domaine de l'électronique numérique pour l'enseignement, notamment ceux qui permettent la création de schémas électroniques. Nous distinguerons ceux qui permettent d'effectuer des évaluations des apprenants par rapports à ceux qui ne le font pas.

Ensuite, nous présenterons la plate-forme TEB (pour Travaux pratiques d'Électronique Binaire) que nous avons développée dans le but de gérer et d'évaluer des schémas électroniques des apprenants.


## 3.1 Quelques outils et logiciels

Pour cet état de l'art, nous avons choisi des outils ou logiciels qui sont libres d'accès (Shareware[1] ou Freeware[2] par exemple) ou Open Source[3] afin de pouvoir les tester et les exécuter pour connaître leurs fonctionnalités. Nous avons donc sélectionné une liste d'outils ou de logiciels qui ont les caractéristiques suivantes :

- disponibilité d'une version complète ou de démonstration de l'outil ;
- outil à visée pédagogique ;
- simple à utiliser et à manipuler ;
- disposant d'une interface graphique pour la création de schémas électroniques ;
- possibilité de créer et d'exécuter des simulations ;
- possibilité de récupérer les schémas créés dans des fichiers à part ;
- éventuellement, possibilité de faire des évaluations sur les schémas ou sur les connaissances des apprenants.

Dans les sections qui vont suivre, nous allons présenter ces différents outils.

### 3.1.1 Automation Studio

*Automation Studio*™[4] (figure 3.1) est une solution complète et intégrée, qui permet de concevoir, simuler et animer des circuits dans divers contextes applicatifs : hydraulique, pneumatique, électrique, électronique, etc. C'est un outil pédagogique destiné aux enseignants et aux apprenants. Il permet d'illustrer les concepts étudiés en cours grâce à la simulation et à l'animation.

[1] Version complète d'un logiciel afin de le tester sans risque. Si on décide de l'utiliser régulièrement, on devra s'acquitter du prix de la licence.

[2] Version gratuite d'un logiciel que l'on peut utiliser et distribuer librement, mais qui reste la propriété de son auteur.

[3] Code source libre.

[4] http ://www.automationstudio.com/EDUC/fr/index.htm

Par ailleurs, aucune fonction d'évaluation n'est incluse dans l'outil et le format des fichiers qui contiennent les schémas créés est binaire. Il n'y donc aucun moyen de les interpréter pour les analyser ou les évaluer. De plus, la diversité des domaines traités par l'outil pourrait perturber la compréhension de l'apprenant et donc son apprentissage. Finalement, ce logiciel est soumis à une licence d'utilisation (Shareware).

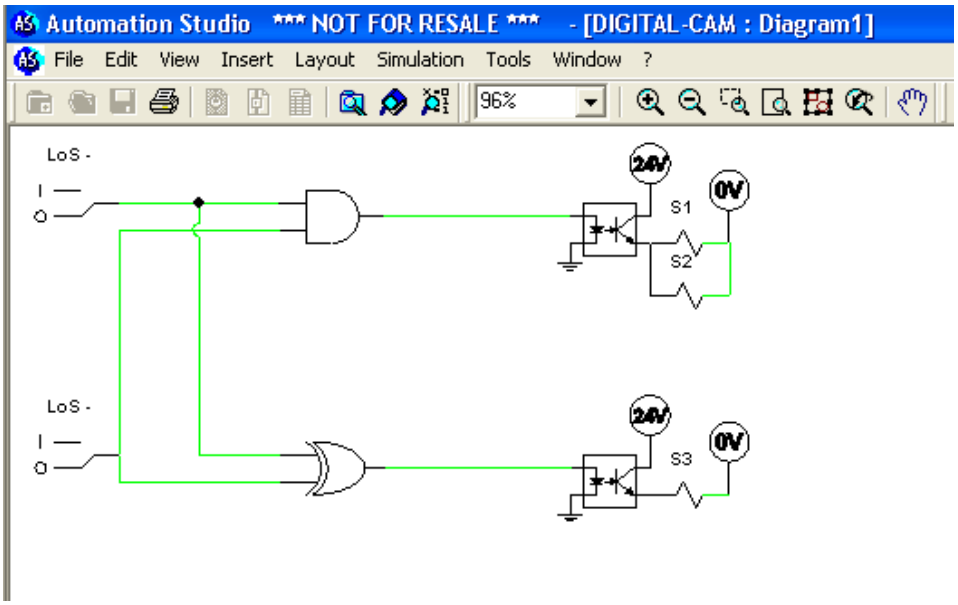



FIG. 3.1 – Interface de *Automation Studio*™

### 3.1.2 Circuit Builder

*Circuit Builder*[5] (figure 3.2) est une simple « applet » Web pour la création et la simulation de schémas électroniques élémentaires. Les fonctionnalités sont très simples d'utilisation mais très limitées. Il permet de créer des schémas électroniques à partir de certains composants de base et affiche automatiquement la table de vérité [6] correspondante.

Par ailleurs, il lui manque certains opérateurs qui sont souvent utilisés, comme le *xor* par exemple, et ne propose que des opérateurs ayant deux « entrées »[7], ce qui pourrait limiter le pouvoir d'expression des apprenants. De plus, comme c'est une applet, il est impossible de sauvegarder les schémas pour les réutiliser. Enfin, comme le précédent, il ne propose aucune évaluation.

[5] http ://www.jhu.edu/v̄irtlab/logic-circuits/
[6] cf. Chapitre 4 pour la définition de la table de vérité.
[7] L'arité des composants est égale à 2.

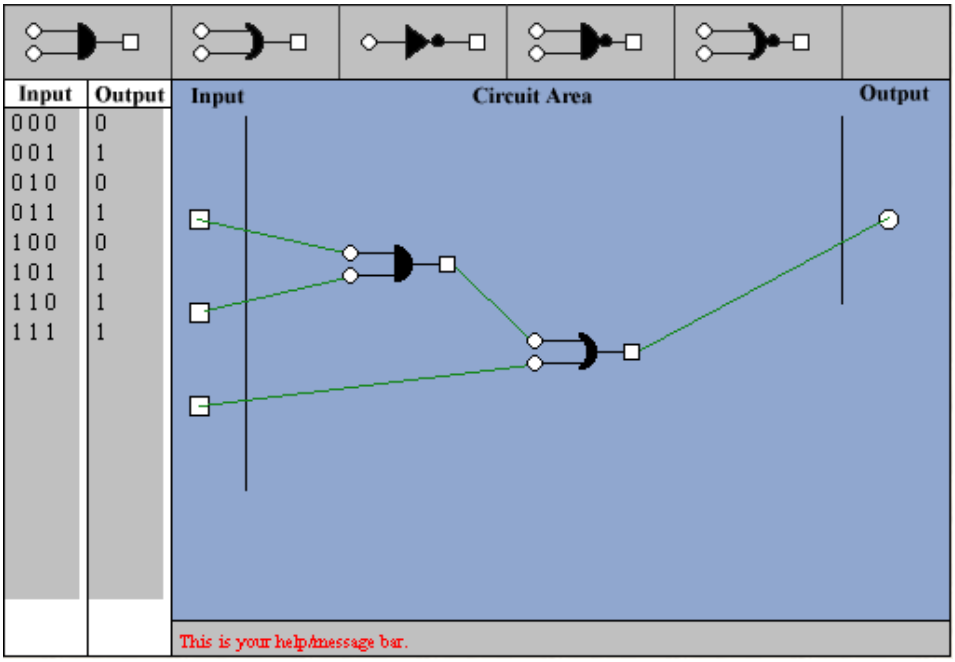



FIG. 3.2 – Interface Web de *Circuit Builder*

### 3.1.3 DigSim

*DigSim*[8] (figure 3.3) est un simulateur Open Source de circuits électroniques écrit en Java. Il permet de créer et simuler des circuits électriques et électroniques logiques. Le logiciel est assez complet et contient beaucoup de fonctionnalités. Il peut visualiser les états logiques des sorties du circuit ou tracer les chronogrammes[9] de fonctionnement. La dernière version de DigSim a la particularité en plus d'intégrer quelques applications commandées par des circuits, par exemple : une perceuse, un chariot, un feux de carrefour, etc. Elle permet aussi la simulation de circuits plus complexes comme les mémoires RAM ou PROM.

Parmi les inconvénients de DigSim, nous distinguons le format « binaire » des fichiers contenant les schémas des circuits . Ensuite, comme pour Automation Studio, il est très complet et risque de désorienter l'apprenant dans son apprentissage et enfin, il ne dispose d'aucun système d'évaluation des schémas des apprenants.

[8] http ://patrick.furon.free.fr/_elecnumerique/_cours_electronum/_DigSimLancementDemo.html
[9] Courbe graphique qui illustre l'état des sorties d'un circuit électronique en fonction du temps.

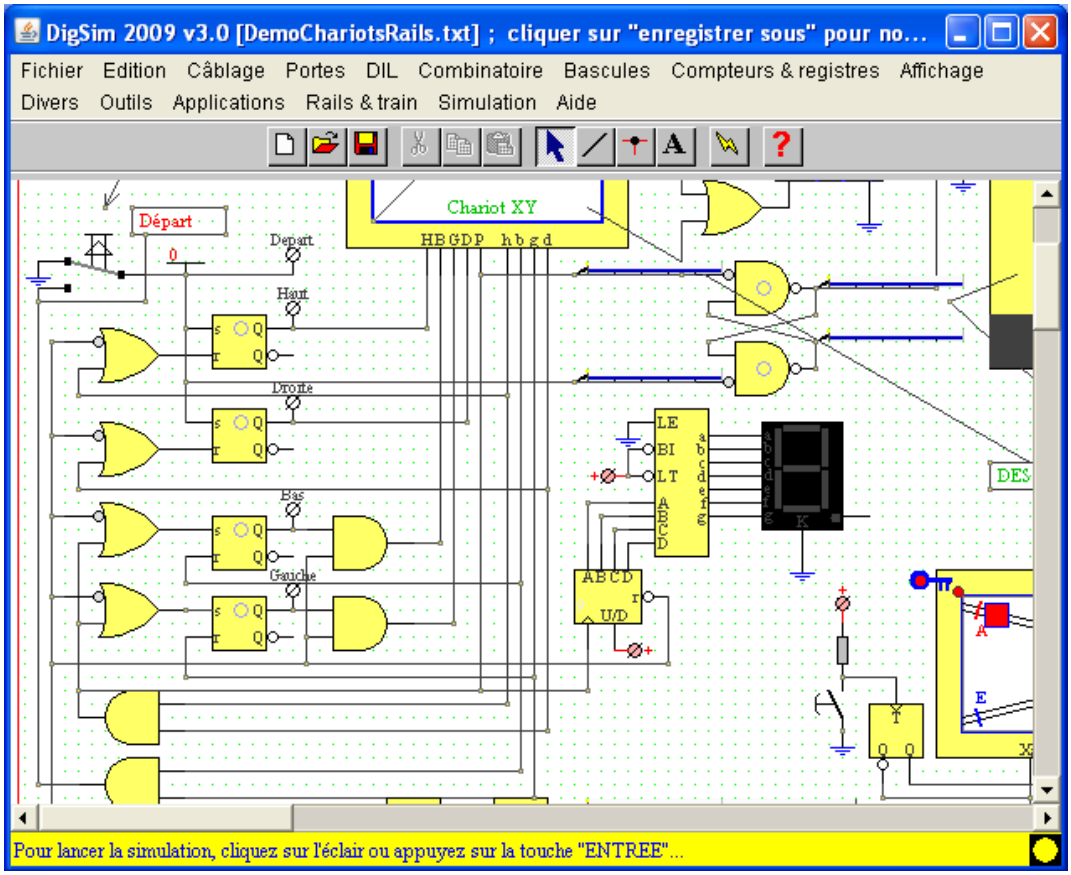



FIG. 3.3 – Interface du simulateur *DigSim*

### 3.1.4 LogicSim

*LogicSim*[10] (figure 3.4) est une application Open Source de simulation électronique numérique développée en Java. Très simple à utiliser, elle contient l'essentiel pour créer et simuler des circuits logiques pédagogiques.

Le format des schémas créés est « binaire », il est donc inutilisable en dehors de cet environnement. L'application ne permet aucune évaluation des circuits électroniques. De plus, les composants proposés disposent uniquement de deux « entrées »[11].

[10] http ://www.tetzl.de/java_logic_simulator.html
[11] L'arité des composants est égale à 2.

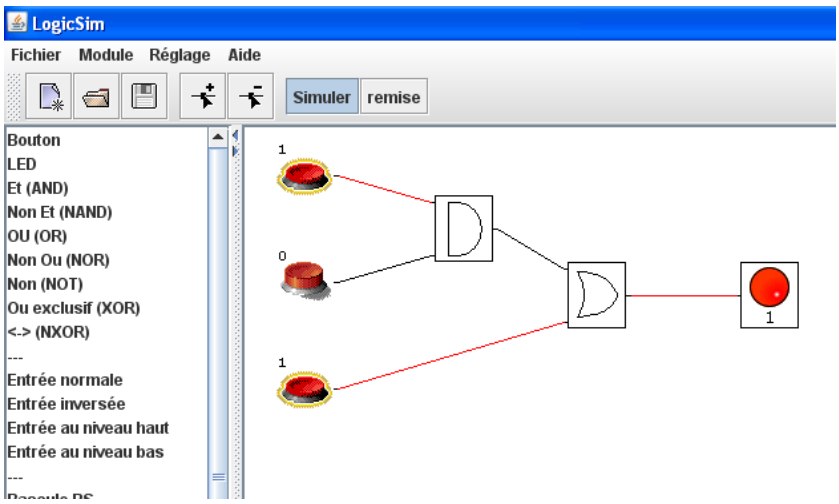



FIG. 3.4 – Interface de l'outil *LogicSim*

### 3.1.5 LogiSim

*LogiSim*[12] (Burch, 2002) (figure 3.5) est un outil Open Source destiné à la conception et la simulation de circuits électroniques pédagogiques. Il dispose d'une interface graphique assez intuitive et facile d'utilisation. LogiSim est aussi conçu pour la création et la simulation d'une Unité Centrale de Processeur (*CPU* en anglais) à visée pédagogique. Un des avantages de cet outil est qu'il permet de sauvegarder les circuits créés dans un format standard XML (*Extensible Markup Language* en anglais). Hormis la simulation, aucune autre option, telle que l'évaluation par exemple, n'est disponible dans cet environnement.

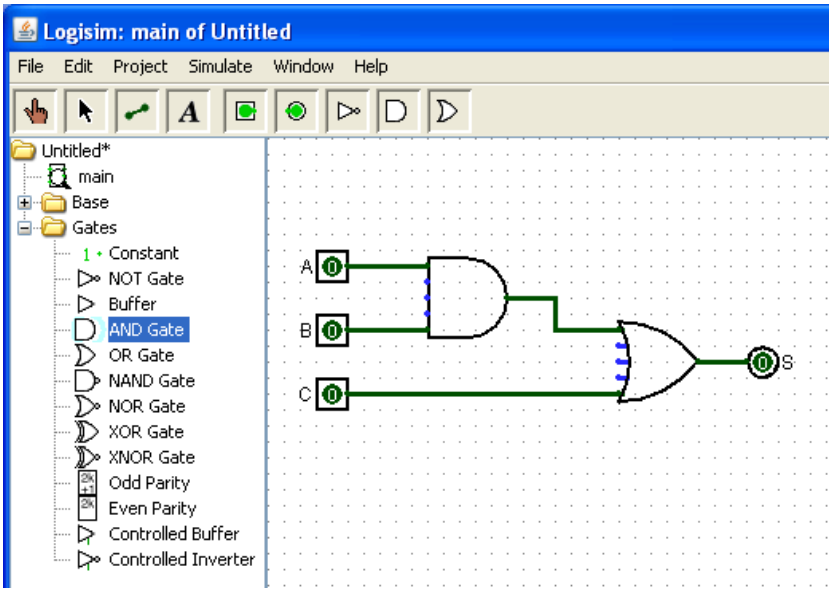



FIG. 3.5 – Interface de l'outil *LogiSim*

[12] http ://ozark.hendrix.edu/burch/logisim/

### 3.1.6 MMLogic : A MultiMedia Logic Design System

*MMLogic*[13] (figure 3.6) est un logiciel de simulation de circuits électroniques, Open Source et très simple à utiliser. Il dispose d'un format propre pour sauvegarder les schémas créés et ne contient aucune méthode d'évaluation de l'apprenant. À part la possibilité de créer des circuits logiques et de les simuler, aucune autre option n'est disponible pour ce logiciel.

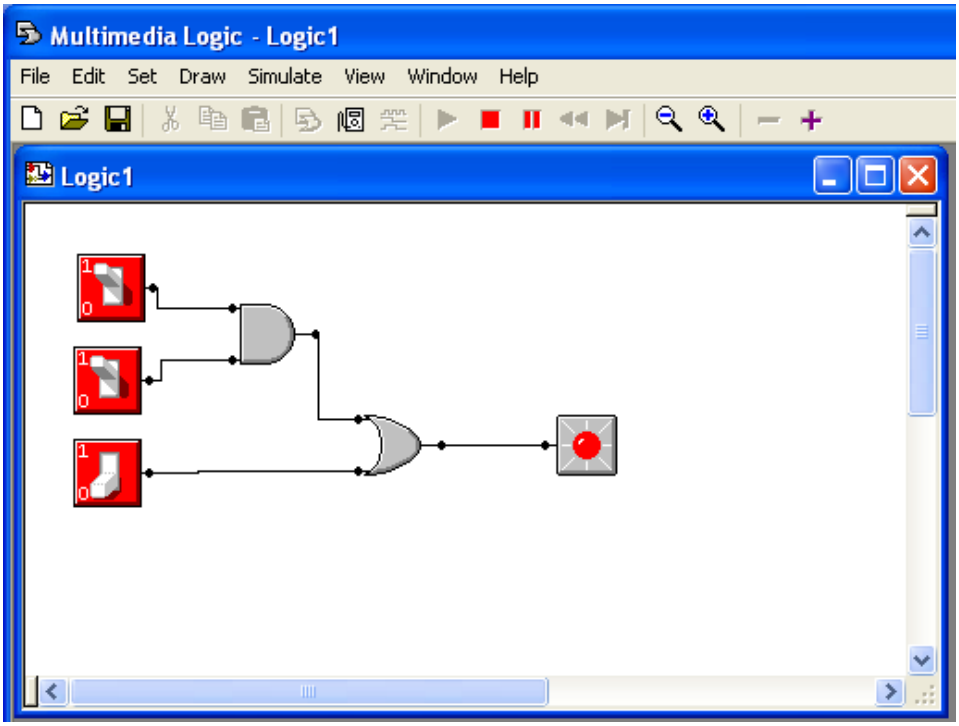



FIG. 3.6 – Interface de l'outil *MMLogic*

### 3.1.7 SimulPortes

*Simulportes*[14] (figure 3.7) est un logiciel de simulation de circuits numériques destiné à l'apprentissage du fonctionnement des opérateurs logiques, du principe des équations logiques et des tables de vérité. En effet, à la création d'un circuit, le logiciel permet d'afficher la table de vérité et l'équation logique correspondantes. À part ça, le logiciel utilise les symboles de la norme IEEE/ANSI 91-1984 pour représenter les composants logiques[15].

L'utilisation du logiciel est simple et intuitive. C'est un outil complémentaire aux cours et travaux pratiques sur les circuits logiques. Par contre, aucun moyen d'évaluation des circuits ou des connaissances des apprenants n'est intégré dans le logiciel. De plus, les composants proposés disposent uniquement de deux « entrées »[16].

[13] http ://www.softronix.com/
[14] http ://www.vbsimulog.fr.st/
[15] Le symbole & représente le composant *And*, le $\geq 1$ représente le composant *Or*, etc.
[16] L'arité des composants est égale à 2.

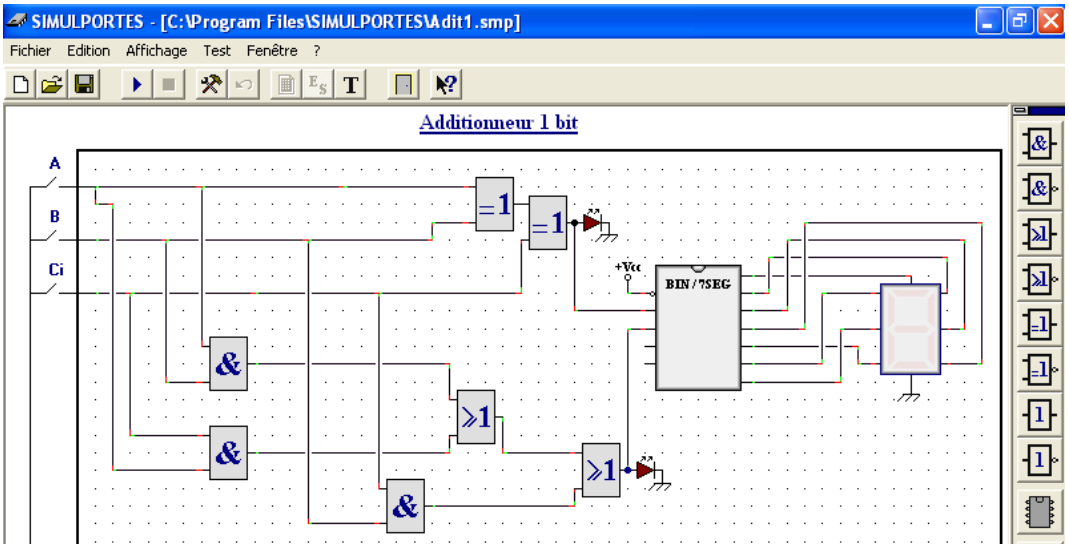



FIG. 3.7 – Interface du logiciel *Simulportes*

### 3.1.8 National Instruments Electronics Workbench : Multisim

*Multisim*[17] de *National Instruments* (figure 3.8) est une plate-forme intégrée dédiée à la conception et le test de circuits électroniques. Il permet la création et la simulation SPICE[18] de schémas électroniques (numériques et analogiques), d'étudier ces schémas en utilisant des analyses SPICE avancées, de construire leurs montages dans un environnement 3D afin de vérifier certaines contraintes ou limitations du circuit, avant de passer finalement à son prototypage réel.

Le logiciel *Multisim* a été conçu pour répondre aux besoins des enseignants en matière de conception et simulation de circuits électroniques, et d'accompagner les apprenants dans leur apprentissage des bases sur les circuits électroniques. Il dispose d'une bibliothèque de composants SPICE assez complète et intègre des Quizz[19] pour l'évaluation des connaissances des apprenants.

Par contre, cette plate-forme est soumise à une licence d'utilisation non gratuite, elle dispose d'un format propre pour les schémas des circuits et nécessite un minimum de formation pour une utilisation efficace, aussi bien pour l'enseignant que pour les apprenants.

[17] http ://www.ni.com/multisim/f/

[18] SPICE est un standard industriel qui donne accès à un grand nombre de bibliothèques de modèles de composants développés par les fabriquants.

[19] Questionnaire portant sur des connaissances particulières.

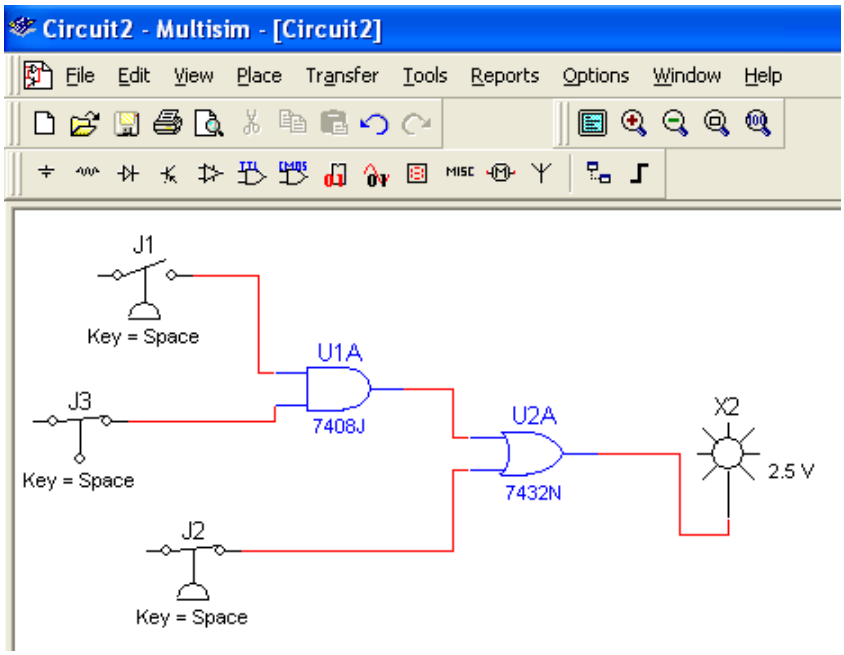



FIG. 3.8 – Interface du logiciel *Multisim*

### 3.1.9 L'électronique en question

*L'électronique en question*[20] est un cédérom pour accompagner la formation des apprenants en électronique. Les objectifs pédagogiques du cédérom sont les suivants : *offrir à l'élève les moyens d'établir des liens entre les différents domaines de l'électronique et entre l'électronique et les disciplines connexes* (Lemarchand *et al.*, 2000). Le produit a la possibilité de lancer quelques simulations guidées, c'est-à-dire, utilisant les seules options proposées par l'outil. Par contre, il n'offre pas la possibilité de créer des schémas de circuits.

Si nous présentons cet outil parmi les produits précédents, c'est juste pour indiquer qu'il offre la possibilité d'évaluer les connaissances des apprenants sous format de questionnaires ou d'exercices simples.

### 3.1.10 Conclusion

Finalement, nous allons synthétiser l'analyse précédente dans le tableau 3.1.

[20] http ://www.en-questions.net/bel/bel.htm

| Nom | Simple | Complet | Arité≥ 2 | Open Source | Table de vérité | Format standard | Évaluation |
|---|---|---|---|---|---|---|---|
| **Automation Studio** | | x | x | | | | |
| **Circuit Builder** | x | | | | | | |
| **DigSim** | | x | x | x | x | | |
| **LogicSim** | x | | | x | | | |
| **LogiSim** | x | x | x | x | | x | |
| **MMLogic** | x | x | | x | | | |
| **SimulPortes** | x | x | | | x | | |
| **Multisim** | | x | | | | | x |
| **L'électronique en question** | x | | | | | | x |

TAB. 3.1 – Récapitulatif des logiciels d'électronique numérique avec certaines de leurs caractéristiques

Les différents critères utilisés pour cette synthèse sont expliqués dans la liste suivante :

- « Simple » : ce critère désigne si le logiciel est simple d'utilisation ;
- « Complet » : ce critère désigne si le logiciel contient toutes les fonctionnalités de base pour créer des circuits électroniques ;
- « Arité≥ 2 » : ce critère désigne si le logiciel comporte des opérateurs logiques ayant des arités supérieures à 2 ;
- « Open Source » : ce critère désigne si le logiciel est libre d'utilisation ;
- « Table de vérité » : ce critère désigne si le logiciel affiche la table de vérité ;
- « Format standard » : ce critère désigne si le logiciel sauvegarde les circuits dans des fichiers ayant un format réutilisable ;
- « Évaluation » : ce critère désigne si le logiciel propose des évaluations.

Nous remarquons que parmi tous les outils et logiciels décrits dans ce tableau, la plupart sont simples d'utilisation, permettent d'effectuer des simulations de circuits électroniques, récupèrent les schémas des circuits dans des fichiers ayant un format binaire et aucun n'effectue une évaluation des circuits créés.

Nous avons tout de même retenu un logiciel particulier *LogiSim* (Burch, 2002) qui permet en plus, par rapport aux autres logiciels, d'enregistrer les circuits électroniques dans un format standard XML. Nous verrons dans la section suivante l'utilité d'un tel logiciel.

## 3.2 La plate-forme TEB : Travaux pratiques d'Électronique Binaire

Pour le besoin de nos travaux de recherche sur l'évaluation de l'apprenant, nous avons été amené à développer une plate-forme Web destinée à la gestion et l'évaluation de circuits

électroniques produits par les apprenants (Tanana *et al.*, 2006, 2008a). Cette plate-forme propose une « aide à la correction » destiné à faciliter la tâche d'évaluation de l'apprenant à l'enseignant. Elle pourra s'intégrer, par la suite, dans une activité d'évaluation d'un outil plus complet.

Nous allons décrire dans cette section les différentes fonctionnalités et composantes de cette plate-forme.

### 3.2.1 Description

L'application *TEB* (figure 3.9) est une plate-forme Web dédiée à la gestion de travaux pratiques d'électronique numérique (Tanana *et al.*, 2006, 2008a). Cette application est destinée à trois catégories d'utilisateur :

- l'enseignant, pour proposer des sujets de travaux pratiques et pour évaluer des schémas de circuits électroniques réalisés par les apprenants, même en grand nombre ;
- aux apprenants, pour concevoir et réaliser, à distance ou en présentiel, des schémas de circuits électroniques numériques ;
- et aux administrateurs, pour la gestion des comptes utilisateurs (enseignant et apprenant).


FIG. 3.9 – Interface de la plate-forme Web *TEB*

La plate-forme utilise le simulateur Open Source *Logisim* (Burch, 2002) qui permet de créer et de tester des schémas de circuits électroniques. Dans les paragraphes suivants, nous allons décrire le fonctionnement de la plate-forme.

L'enseignant commence par fournir un exercice à l'apprenant, constitué d'un énoncé, éventuellement accompagné d'un système d'aide en ligne (documentation technique, rappel de cours, formulaires, schémas, etc.) et d'une liste de critères d'évaluation (par exemple, les résultats des sorties du circuit, le nombre de composants ou connexions utilisés, le temps mis à la réalisation du circuit, etc.). Chaque critère est accompagné d'un barème et d'un « pas de décrémentation » en cas d'erreur. Un schéma de référence du circuit est associé, par l'enseignant, à chaque exercice. Il permet de récupérer la table de vérité du circuit à réaliser.

L'apprenant récupère le sujet de l'exercice, consulte éventuellement les différents documents attachés et crée un schéma conforme aux exigences demandées par l'énoncé à l'aide du même simulateur. Il peut aussi consulter les différents critères d'évaluation et leurs barèmes. Une fois le schéma réalisé et testé, l'apprenant envoie à travers la plate-forme le résultat final à l'enseignant.

L'enseignant récupère les travaux des différents apprenants et lance le « moteur d'évaluation ». Ce moteur permet à l'enseignant d'effectuer une comparaison automatique entre son schéma de référence et tous ceux réalisés par les apprenants. Le moteur d'évaluation compare les deux tables de vérité « apprenant-enseignant » en se servant du simulateur *Logisim* et en se basant sur les critères/barèmes définis préalablement. Les résultats des évaluations de tous les apprenants sont ensuite affichés sous format de notes /20, ce qui permet d'effectuer une première **évaluation sommative**. Dans tous les cas, l'enseignant a la possibilité de revoir directement les schémas des apprenants pour une évaluation plus approfondie.

La note obtenue, est calculée en fonction :

- de la note maximale (par exemple : 20/20) ;
- des critères qui sont sélectionnés par l'enseignant (par exemple : les valeurs des sorties en fonction des entrées) ;
- des barèmes associés (par exemple : $-1$ pour chaque 2 erreurs sur une des sorties).

Le fonctionnement actuel du moteur d'évaluation est schématisé par la figure 3.10.

La méthode d'évaluation utilisée par le moteur de la plate-forme n'est pas tout à fait « adaptée ». En effet, comme nous le verrons dans le chapitre 4, la création d'un schéma électronique correspond à toute une démarche de résolution et la simple comparaison des tables de vérité des schémas électroniques de l'enseignant et de l'apprenant n'est pas toujours représentative du résultat fourni par ce dernier.

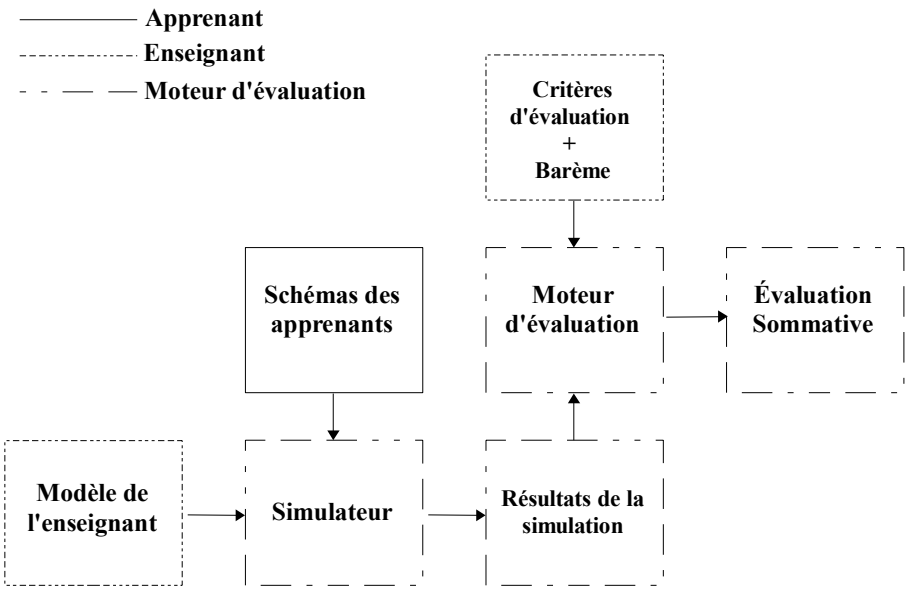



FIG. 3.10 – Fonctionnement actuel de la plate-forme *TEB*

### 3.2.2 Expérimentation de la plate-forme

Cette version de la plate-forme a été expérimentée en situation réelle pour des travaux pratiques en électronique numérique, auprès des élèves de 3e année de l'ENSA de Tanger[21] (Tanana *et al.*, 2006). Elle s'est déroulée en deux séances présentielles consécutives, par groupe de 12 binômes ou trinômes, sous la surveillance de deux enseignants spécialistes en électronique et informatique respectivement.

Le but de cette première expérimentation n'étant pas l'évaluation de la plate-forme elle même, mais la production de schémas électroniques afin d'évaluer le « moteur d'évaluation ». Quatre exercices ont été ainsi proposés aux élèves, ce qui a permis de récupérer au total quatre séries de 24 schémas électroniques par exercice.

Ensuite, les enseignants ont procédés à l'exécution du moteur d'évaluation disponible sur la plate-forme ce qui a permis d'avoir une évaluation rapide de tous les circuits des apprenants, sous format de note sur 20. Cette version prend en compte des critères d'évaluation quantitatifs qui concernent uniquement la réponse du circuit lorsqu'il est soumis à une combinaison binaire des entrées.

Ce que nous avons constaté pour cette première expérimentation, c'est que, malgré la rapidité avec laquelle nous obtenons la note finale, cette dernière est rarement représentative des connaissances de l'apprenant. En effet, nous avons remarqué que tous les circuits qui ont obtenu une note maximale de 20/20 (aucune erreur) n'étaient pas tous identiques du point de vue « schéma électronique », certains étaient plus simplifiés que d'autres et donc plus

[21] Ecole Nationale des Sciences Appliquées de Tanger - Maroc

optimisés. Dans d'autres cas, quelques erreurs au niveau d'une seule sortie du circuit (dans le cas de sorties multiples) pouvaient pénaliser la note finale de l'ensemble du circuit.

Finalement, le « moteur d'évaluation » est incapable de décider à quelle étape de résolution de l'exercice s'est produite l'erreur. Par exemple, le fait que l'élève oublie de créer un lien entre deux composants, sera considéré par la correction de l'enseignant comme une erreur de construction du schéma électronique. Le moteur d'évaluation va considérer cet oubli comme une erreur au niveau de la table de vérité.

### 3.2.3 Conclusion

Dans ce chapitre, nous avons décrit le fonctionnement d'une plate-forme pour la gestion des schémas électroniques. Cette plate-forme utilise un logiciel de simulation et permet d'effectuer des évaluations sommatives basées sur la comparaison des tables de vérité des schémas de l'enseignant et de l'apprenant respectivement. Nous nous sommes rendu compte que cette manière d'évaluer n'était pas suffisante pour avoir une évaluation correcte. En effet, la démarche pour créer un schéma électronique est compliquée et mérite donc d'être approfondie.

Nous avons finalement décidé d'étudier la démarche de réalisation d'un schéma électronique. Ensuite, nous entamerons une nouvelle réflexion sur la manière d'effectuer une évaluation formative de ces schémas. Plus précisément, nous allons nous intéresser à l'évaluation de la forme du schéma électronique produit par l'apprenant.

Par la suite, le fonctionnement souhaité du moteur d'évaluation sera schématisé par la figure suivante :

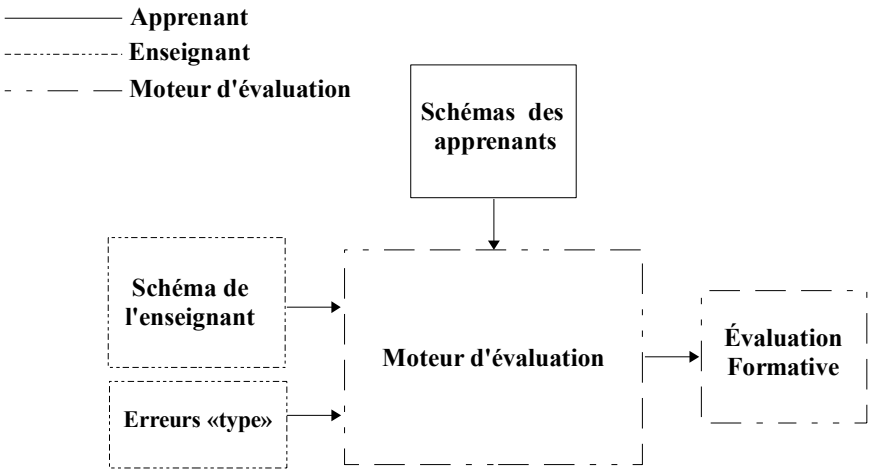



FIG. 3.11 – Fonctionnement souhaité de la plate-forme *TEB*

## Deuxième partie

# Évaluation formative du savoir-faire de l'apprenant

# Chapitre 4

# Synthèse des circuits logiques combinatoires

**Sommaire**

## Résumé

Le domaine de l'électronique numérique s'est généralisé depuis qu'on a commencé à s'intéresser à la numérisation de l'ensemble de la chaîne d'information. Malgré la complexité des circuits numériques, les principes fondamentaux sur lesquels ils reposent sont étonnamment simples. Les apprenants, pendant leur apprentissage, doivent comprendre les mécanismes de base nécessaires à la construction et l'exploitation des systèmes numériques de traitement. Pour cette raison, la plupart des écoles d'ingénieurs proposent des cours en rapport avec le traitement numérique de l'information. Ces cours nécessitent des pré-requis, comme par exemple la synthèse de circuits logiques, qui sont généralement proposés et enseignés en début du cycle d'ingénieurs.

Dans ce chapitre, nous commencerons par quelques définitions des circuits logiques. Ensuite, nous présenterons quelques méthodes de représentation et de simplification des fonctions logiques. Nous définirons la synthèse d'un circuit logique, illustrée par un exemple. Nous proposerons une représentation abstraite des schémas électroniques et nous finirons par présenter brièvement les circuits logiques manipulés et évalués dans la plate-forme TEB.

## 4.1 Définition d'un circuit logique

Un *circuit logique* est la définition d'un circuit électronique réalisant une ou plusieurs fonctions. Il est composé d'un ensemble de portes logiques interconnectées entre-elles.

Une *porte logique* est un circuit électronique élémentaire réalisant une opération simple. Nous pouvons dénombrer des portes logiques de base :

- *not*, qui représente la négation ;
- *and*, qui représente le *et logique* ;
- *or*, qui représente le *ou logique* ;

et des portes logiques composées, construites à partir d'une combinaison des trois portes logiques de base :

- *nand*, qui représente la négation du *et logique* ;
- *nor*, qui représente la négation du *ou logique* ;
- *xor*, qui représente le *ou exclusif* ;
- *nxor*, qui représente la négation du *ou exclusif*.

Une porte logique peut être constituée de plusieurs entrées et une seule sortie.

Un *circuit logique « combinatoire »* est un circuit logique qui possède $n$ entrées $E_{i(1\leq i\leq n)}$ et $m$ sorties $S_{j(1\leq j\leq m)}$, de telle sorte que $S_j = f(E_1, \ldots, E_n)$. $f$ est appelée « *fonction logique* » et dépend uniquement de l'état des entrées $E_i$.

Les fonctions logiques de base sont issues des mathématiques théoriques de l'algèbre de Boole, qui est une méthode mathématique déductive créée par le mathématicien anglais

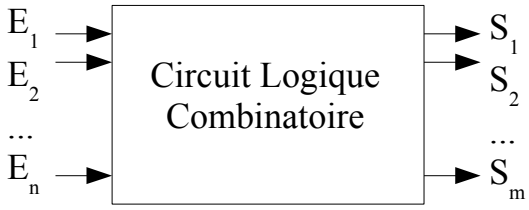



FIG. 4.1 – Modèle d'un circuit logique combinatoire

George BOOLE dans son ouvrage *The Laws of Thought* (1854). Ensuite, c'est Claude SHANNON (1938) qui fut le premier à faire le rapprochement entre les nombres binaires, l'algèbre de Boole et les circuits électriques. Il démontre que le système binaire manipulé par les opérations logiques de Boole (0=Faux, 1=Vrai) permet de réaliser toutes les opérations logiques et arithmétiques, et définit la quantité d'information élémentaire : le *bit* (BInary digiT).

L'algèbre de Boole permet donc de traduire des signaux électriques en expressions mathématiques plus faciles à manipuler par les informaticiens. Elle se caractérise par (Darche, 2002) :

- un ensemble à deux éléments $\{0, 1\}$ ;
- une opération unaire involutive $\bar{}$ (la négation) ;
- deux lois de composition internes : **.** et **+**, commutatives, associatives, ayant un élément neutre et distributives l'une par rapport à l'autre.

La loi **.** est représentée par l'opérateur *and*, la loi **+** est représentée par l'opérateur *or* et la négation par l'opérateur *not*.

Par conséquent, les variables de l'algèbre booléenne, appelées *variables logiques*, ne peuvent avoir qu'une des deux valeurs 0 ou 1. Elles peuvent être réunies à l'aide des opérateurs élémentaires pour former des *expressions logiques*.

## 4.2 Représentation d'une fonction logique

Les fonctions logiques sont représentées algébriquement à l'aide des expressions logiques construites à partir des opérateurs élémentaires $\{not, and, or\}$ (Nketsa, 1998) ou des opérateurs composés $\{nand, nor, xor, nxor\}$. En dehors de cette représentation algébrique, il existe d'autres méthodes permettant de les représenter. Dans les sections suivantes, nous allons présenter certaines de ces méthodes.

### 4.2.1 Méthode algébrique

Une fonction logique est représentée par une expression algébrique sous la forme de la somme des termes « produit », appelée aussi *première forme canonique* (la forme la plus courante) ou sous la forme de produit des termes « somme », appelée aussi *deuxième forme canonique* (Darche, 2002; Nketsa, 1998). Les termes de la première forme canonique représentent les cas où la fonction vaut « 1 ». Ils sont appelés des « mintermes ». Chaque « minterme » comporte toutes les variables d'entrée sous forme « vraie » ou « inverse ».

La figure 4.2 représente un exemple d'une fonction logique représentée sous la forme de la somme des termes produit. Cette fonction s'appelle « $f$ » et représente un circuit logique à 3 entrées et une seule sortie.

$$f(a,b,c) = \overline{a}.\overline{b}.c + a.\overline{b}.\overline{c} + a.\overline{b}.c + a.b.\overline{c} + a.b.c$$

FIG. 4.2 – La première forme canonique de la fonction logique « $f$ »

La même fonction, après avoir été simplifiée (comme nous le verrons plus loin dans la section 4.3), sera représentée par l'expression logique suivante :

$$f(a,b,c) = a + \overline{b}.c$$

FIG. 4.3 – La forme simplifiée de la fonction logique « $f$ »

Ce mode de représentation n'est pas limité par le nombre de variables d'entrée, mais il devient très difficile à gérer lorsque ce nombre augmente.

### 4.2.2 Méthodes tabulaires

Une fonction logique peut aussi être représentée par un tableau. Il existe plusieurs représentations tabulaires des fonctions logiques, nous allons détailler les plus connues d'entre elles.

#### 4.2.2.1 Table de vérité

Pour déduire une fonction logique, il est en général intéressant d'utiliser un tableau renseignant les états de chaque sortie en fonction de tous les états possibles des entrées. Ce tableau porte le nom de *Table de Vérité*.

La table de vérité d'un circuit à $n$ entrées a autant de lignes que d'états d'entrée possibles, soit $2^n$. Pour chacun de ces états, la sortie peut prendre la valeur 0 ou 1. L'ordre des lignes n'est pas important, mais il correspond généralement à l'ordre croissant du nombre binaire formé par les variables d'entrée.

| a b c | S |
|---|---|
| 0 0 0 | 0 |
| 0 0 1 | 1 |
| 0 1 0 | 0 |
| 0 1 1 | 0 |
| 1 0 0 | 1 |
| 1 0 1 | 1 |
| 1 1 0 | 1 |
| 1 1 1 | 1 |

TAB. 4.1 – Table de vérité de la fonction logique « $f$ » de la figure : 4.2

Chaque table de vérité correspond à une et une seule fonction logique. Pour $n$ variables d'entrée différentes, nous pouvons définir $2^{2^n}$ fonctions logiques différentes (Serrell, 1953) et autant de tables de vérité.

#### 4.2.2.2 Table de Karnaugh

La table de Karnaugh (1953) est un outil graphique semblable à la table de vérité. C'est un tableau de deux dimensions comportant $2^n$ cases, où $n$ est le nombre d'entrées. L'ordre des cases correspond au code Gray[1]. Un seul bit de la variable d'entrée change de valeur entre deux cases adjacentes (Darche, 2002).

Comme la table de vérité, la table de Karnaugh met en évidence le rapport entre les entrées et les sorties. Chaque case du tableau, correspond à une ligne de la table de vérité (Nketsa, 1998). Les variables logiques sont réparties sur le tableau verticalement et horizontalement. Les barres sur chaque variable représentent les cases où cette variable prend la valeur logique « 1 ».

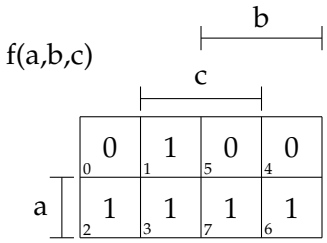



FIG. 4.4 – Tableau de Karnaugh de la fonction « $f$ » de la figure 4.2

Ce mode de représentation est limité à 5 ou 6 variables d'entrée maximum. Il est surtout utile pédagogiquement pour s'initier aux circuits logiques, comme le fait remarquer son inventeur (Karnaugh, 1953).

[1] Le nom du code vient de l'ingénieur américain Frank Gray qui déposa un brevet sur ce code en 1953. Le principe consiste à ne faire changer qu'un seul bit à la fois quand un nombre est augmenté d'une unité.

### 4.2.3 Langages de description

Les langages de description matériel (HDL pour Hardware Description Language) sont des langages de modélisation et de conception informatique qui permettent d'effectuer une description formelle des circuits électroniques. Ils peuvent décrire et simuler le comportement d'un système électronique numérique à tous les niveaux de sa conception. Les langages HDL les plus connus sont : VHDL[2], Verilog[3] et SystemC[4].

```
entity f  is
        port ( a, b, c : in bit; S : out bit)
end f;

architecture comportementale of f is
begin
  process
    if (not a and not b and c) or (a and not b and not c) or
       (a and not b and c) or (a and b and not c) or (a and b and c)
              S <= '0'
    else
              S <= '1'
    end if;
    wait on a, b, c;
  end process;
end comportementale
```

FIG. 4.5 – Extrait du code VHDL pour la représentation de la fonction logique « $f$ » de la figure 4.2

La particularité de ces langages est qu'ils permettent la modélisation de systèmes complexes pouvant être intégrés dans des microprocesseurs ou dans des circuits spécifiques. Ils demandent donc un apprentissage préalable et sont destinés généralement à un public spécialisé dans le domaine.

### 4.2.4 Méthodes graphiques

Une fonction logique peut être représentée, sur le plan graphique par un schéma électronique communément appelé : *logigramme*, et sur le plan matériel à l'aide de composants électroniques dont les signaux d'entrées et sorties peuvent prendre des valeurs de l'ordre de 5 volts pour représenter le bit 1, et 0 volts pour représenter le bit 0 (Darche, 2002; Nketsa, 1998).

#### 4.2.4.1 Logigramme

Un logigramme est un schéma graphique normalisé illustrant l'expression d'une fonction logique sans tenir compte des constituants technologiques. C'est à la fois une représen-

[2] http ://www.vhdl.fr/
[3] http ://www.verilog.com/
[4] http ://www.systemc.org/

tation graphique et symbolique qui s'effectue à l'aide des représentations symboliques des opérateurs logiques reliés entre eux. Un logigramme est l'aboutissement final qui permettra de construire techniquement le circuit étudié.

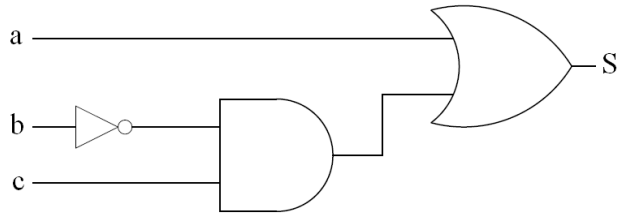



FIG. 4.6 – Un logigramme de la fonction logique « $f$ » de la figure 4.3

#### 4.2.4.2 Diagramme de Décision Binaire ou BDD

Un BDD est une représentation de fonctions logiques basée sur le théorème d'expansion de SHANNON (1938). Les BDDs ont été introduits par S.B. AKERS (1978) et Randal E. BRYANT (1986). Le BDD est une structure de données simple qui s'est avérée très efficace dans la vérification et la manipulation de fonctions logiques. Il permet de représenter de façon canonique et compacte les fonctions logiques, les rendant plus faciles à vérifier.

Un BDD est un graphe orienté acyclique représentant une fonction logique. Il est construit par simplification de l'arbre binaire représentant la décomposition de SHANNON d'une fonction booléenne (Shannon, 1938). La figure 4.7 illustre un exemple d'une fonction logique représentée par un BDD.

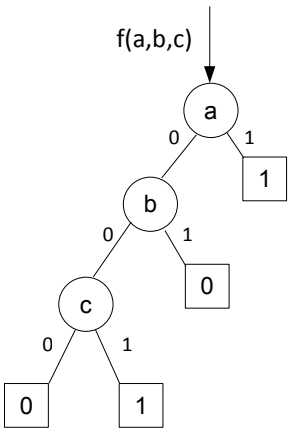



FIG. 4.7 – BDD de la une fonction logique « $f$ » de la figure 4.3

L'intérêt de ce mode de représentation réside plus sur l'aspect de sa structure facilement exploitable par un programme informatique, que sur son aspect graphique.

#### 4.2.4.3 Graphe-Circuit

Une autre façon de représenter une fonction logique est par un « graphe-circuit ». En effet, un circuit logique est constitué d'entrées, de sorties et de plusieurs portes logiques interconnectées. Si nous considérons les entrées, les sorties et les portes logiques comme des nœuds, et les connexions comme des liens, nous pouvons représenter un circuit logique par un graphe. Si en plus le circuit logique est combinatoire (cf. 4.1 page 60), il peut être représenté par un graphe orienté acyclique (Directed Acyclic Graph) (Vollmer, 1999).

Si nous reprenons le logigramme de la figure 4.6, le graphe-circuit correspondant est représenté par la figure 4.8.

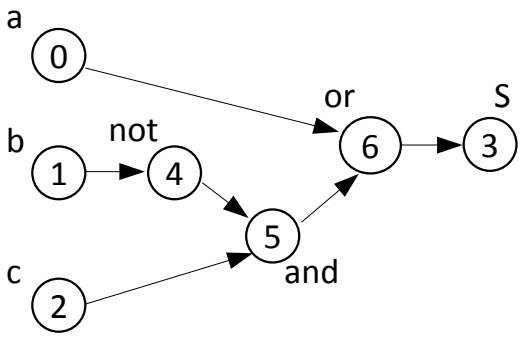



FIG. 4.8 – Graphe-circuit de la fonction logique « $f$ » de la figure 4.3

## 4.3 Simplification d'une fonction logique

Une fonction logique peut être représentée par plusieurs expressions algébriques ou booléennes différentes mais totalement équivalentes. Simplifier une expression logique n'est pas facile car il y a plusieurs solutions possibles. Il faut aboutir à une expression qui minimise le nombre de termes ainsi que le nombre de variables de chaque terme.

Les méthodes de simplification et de conception des circuits logiques que nous utiliserons se basent sur la première forme canonique de la fonction logique (somme de produits). Dès que nous disposons de l'expression algébrique d'un circuit logique, il sera possible de la minimiser pour obtenir une expression comportant moins de termes ou moins de variables par terme. Cette nouvelle équation peut alors servir de modèle pour construire un circuit correspondant entièrement à la fonction logique de la forme canonique, mais qui requiert moins de portes logiques.

Nous allons reprendre la fonction logique « $f$ » de la figure 4.2 page 62, et nous détaillerons les différentes méthodes qui permettent de simplifier cette fonction.

### 4.3.1 Méthode algébrique

L'algèbre de Boole dispose d'un ensemble de lois et théorèmes qui permettent de simplifier de manière performante des expressions algébriques 4.2 (Darche, 2002; Zanella et Ligier, 1998). Ils sont décrits dans le tableau suivant.

| | **Pour l'opérateur** *and* | **Pour l'opérateur** *or* |
|---|---|---|
| **L'élément neutre** | $a.1 = a$ | $a + 0 = a$ |
| **Commutativité** | $a.b = b.a$ | $a + b = b + a$ |
| **Associativité** | $a.(b.c) = (a.b).c$ | $a + (b + c) = (a + b) + c$ |
| **Distributivité** | $a.(b + c) = a.b + a.c$ | $a + (b.c) = (a + b).(a + c)$ |
| **Complémentarité** | $\overline{a}.a = 0$ | $a + \overline{a} = 1$ |
| **Idempotence** | $a.a = a$ | $a + a = a$ |
| **Absorbant** | $a.0 = 0$ | $a + 1 = 1$ |
| **Double négation** | $\overline{\overline{a}} = a$ | |
| **Th. de De Morgan** | $\overline{a.b} = \overline{a} + \overline{b}$ | $\overline{a + b} = \overline{a}.\overline{b}$ |
| **Absorption 1** | $a.(a + b) = a$ | $a + a.b = a$ |
| **Absorption 2** | $a.(\overline{a} + b) = a.b$ | $a + \overline{a}.b = a + b$ |
| **Consensus** | $(a + b)(\overline{a} + c)(b + c) = (a + b)(\overline{a} + c)$ | $a.b + \overline{a}.c + b.c = a.b + \overline{a}.c$ |

TAB. 4.2 – Les principaux théorèmes et lois de l'algèbre booléenne

Les théorèmes de l'algèbre booléenne permettent de simplifier l'expression d'un circuit logique. Par contre, il n'est pas toujours facile de savoir quels théorèmes ou lois utiliser en premier pour obtenir le résultat optimal. Rien ne nous permet non plus de dire que l'expression simplifiée est la « forme optimale » et qu'il n'y a pas d'autres simplifications possibles. Pour toutes ces raisons, la simplification algébrique est souvent un processus d'approximations successives, surtout, si le nombre d'entrées est élevé.

Pour l'exemple de la fonction logique « $f$ » de la figure 4.2 page 62, nous avons appliqué les différentes simplifications algébriques, et nous avons obtenu le résultat suivant :

$$\begin{aligned} f(a,b,c) &= \overline{a}.\overline{b}.c + a.\overline{b}.\overline{c} + a.\overline{b}.c + a.b.\overline{c} + a.b.c \\ &= \overline{a}.\overline{b}.c + a.\overline{b}.\overline{c} + a.\overline{b}.c + a.b.\overline{c} + a.b.c + a.\overline{b}.c \\ &= (\overline{a} + a).\overline{b}.c + a.(\overline{b}.\overline{c} + b.\overline{c} + b.c + \overline{b}.c) \\ &= 1.\overline{b}.c + a.((\overline{b} + b).\overline{c} + (b + \overline{b}).c) \\ &= \overline{b}.c + a.(\overline{c} + c) \\ f(a,b,c) &= a + \overline{b}.c \end{aligned}$$

FIG. 4.9 – Forme simplifiée de la fonction « f »

### 4.3.2 Méthode de Karnaugh

La méthode de simplification d'une fonction par la méthode de Karnaugh s'appuie sur l'adjacence entre les termes de la fonction pour en extraire la représentation la plus simple possible, d'où l'utilisation du code Gray. Cette méthode est bien adaptée à une simplification « manuelle ». Par contre, elle n'est pas facilement implémentable dans un programme informatique et elle est limitée à 5 ou 6 variables d'entrée maximum.

Pour l'exemple de la fonction logique « $f$ » (figure 4.2 page 62), nous obtenons le tableau de Karnaugh suivant :

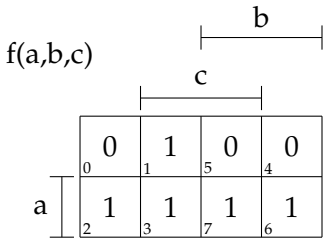



FIG. 4.10 – Tableau de Karnaugh de la fonction « $f$ »

Le principe de cette méthode est de regrouper les cases adjacentes qui sont à 1[5], deux par deux, quatre par quatre ou huit par huit, etc[6]. Ensuite, toutes les cases regroupées sont converties en un « terme logique » contenant les variables de la fonction. Finalement, tous les termes obtenus sont compilés en une somme de termes[7].

Après application de la méthode de Karnaugh pour la simplification de fonctions logiques, nous obtenons le résultat suivant :

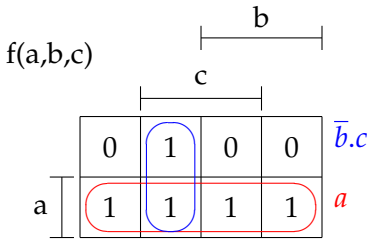



FIG. 4.11 – Simplification de la fonction logique « $f$ » par la méthode de Karnaugh

Ce qui nous donne l'expression logique simplifiée : $S = a + \overline{b}.c$

[5] Si on veut obtenir une « somme de produit », ou à 0, si on veut obtenir un « produit de somme ».
[6] Le nombre de 1 correspond à une puissance de 2.
[7] ou produit de termes si on regroupe les cases à 0.

### 4.3.3 Méthode de Quine-McCluskey

La méthode de Quine-McCluskey est une méthode algorithmique qui permet de contourner la limitation en nombre de variables d'entrée, exigée par la méthode de Karnaugh, et ce afin d'automatiser le processus de simplification d'une fonction logique (Darche, 2002). C'est une technique qui est principalement destinée à une exécution logicielle.

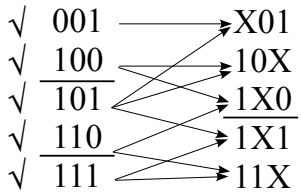
Comme son nom l'indique, elle est issue des résultats des travaux de Quine (1952) et McCluskey (1956). L'implémentation informatique de cette méthode est simple, par contre, elle est compliquée pour une utilisation manuelle.

Nous détaillons les étapes d'exécution de la méthode Quine-McCluskey sur notre exemple de la fonction « $f$ »de la figure 4.2 page 62 :

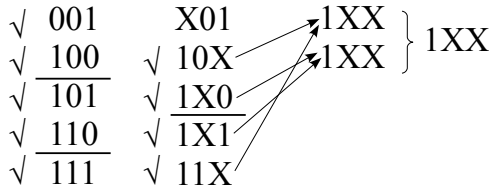
1. Exprimer la fonction sous la forme de la somme des termes « produits » :
$$f(a,b,c) = \overline{a}.\overline{b}.c + a.\overline{b}.\overline{c} + a.\overline{b}.c + a.b.\overline{c} + a.b.c$$
2. Exprimer chaque « minterme » de la fonction sous forme binaire :
$$f(a,b,c) = 001 + 100 + 101 + 110 + 111$$
3. Grouper les termes selon leurs poids[8] :

| **Poids 1** | **Poids 2** | **Poids 3** |
|---|---|---|
| 001 | 101 | 111 |
| 100 | 110 | |
| | | |

4. Unir les termes, ayant un seul bit de différent, deux à deux :




5. Répéter l'étape (4) autant de fois que possible :




6. Identifier les « impliquants premiers » : Un impliquant premier est un terme de la fonction logique qui n'est inclus dans aucun autre terme. Les impliquants premiers sont ceux qui n'ont pas été cochés : X01 et 1XX, ce qui donne respectivement $\overline{b}.c$ et $a$.

[8] Le poids d'un terme est le nombre de bits à 1 dans ce terme.

7. Identifier les impliquants premiers « essentiels » : Un impliquant premier essentiel est un impliquant premier comportant au moins une variable logique ne faisant partie d'aucun autre impliquant premier :

| | 001 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|
| X01 | $\underline{*}$ | | * | | |
| 1XX | | $\underline{*}$ | * | $\underline{*}$ | $\underline{*}$ |

Nous cherchons les colonnes où se retrouve un seul signe, les termes des lignes correspondantes sont des impliquants premiers. Dans notre cas, tous les impliquants premiers sont essentiels.

8. Vérifier si la fonction est entièrement exprimée par ses impliquants essentiels, dans ce cas arrêter l'exécution. Comme tous les impliquants de la fonction $f$ sont ici essentiels, la condition est forcément remplie.

9. Si l'exécution ne s'arrête pas à l'étape (8), choisir les impliquants premiers appropriés. La fonction $f$ est exprimée entièrement par ces impliquants premiers essentiels, le résultat de la méthode Quine-McCluskey est :

$$f(a,b,c) = a + \overline{b}.c$$

## 4.4 Synthèse d'un circuit logique

La synthèse des circuits logiques a pour but la réalisation d'un schéma électronique d'une fonction logique répondant à des spécifications particulières. Lors de cette synthèse, il est souhaitable d'obtenir la forme d'équation équivalente qui produira un circuit de taille minimale. Zanella et Ligier (1998) définissent « la synthèse d'un circuit logique » comme suit :

> *« À partir du grand nombre d'expressions algébriques équivalentes qui correspondent à la fonction, il faut en choisir une et la simplifier. Il faut ensuite choisir un logigramme parmi ceux qui conviennent.*
>
> *La synthèse implique donc le choix délicat d'un bon chemin parmi tous ceux qui conduisent de la fonction logique aux logigrammes qui la réalisent. Le critère retenu est, en général, le coût du circuit obtenu ; le circuit le meilleur étant le plus simple donc le moins coûteux (en portes, connexions, . . .) »*

Nous voyons bien qu'il y a plusieurs solutions possibles toutes aussi « justes », les unes que les autres. Par contre, si nous nous basons sur certains critères (nombre de portes, nombre

de connexions, le degré de simplification de l'expression logique, etc.), nous pourrions classer ces solutions les unes par rapport aux autres.

D'une façon générale, les exercices proposés aux apprenants, demandent de faire la synthèse d'un circuit logique répondant à certaines spécificités. La démarche pédagogique classique pour faire la synthèse d'un circuit logique combinatoire est la suivante (Zanella et Ligier, 1998) :

1. définir le nombre des entrées et sorties du circuit à partir de l'énoncé ;
2. construire la table de vérité à partir de la définition du circuit ;
3. déduire une expression logique du circuit à l'aide des méthodes de simplification : table de Karnaugh, lois de l'algèbre de Boole. Ceci consiste à obtenir une expression logique avec le minimum d'opérateurs logiques, afin de réduire le « coût » d'implémentation du circuit ;
4. réaliser le schéma électronique à l'aide des différents opérateurs : not, and, or, nand, nor, xor ou nxor.

Le schéma obtenu est optimisé si la fonction logique correspondante a été simplifiée de façon minimale. Il peut exister plusieurs solutions.

## 4.5 Exemple de la synthèse d'un circuit logique

Pour cet exemple, nous avons pris l'exercice sur la synthèse d'un « additionneur binaire avec retenue ». C'est un exercice typique et simple, proposé à des élèves de 3$^{e}$ année d'école d'ingénieurs ou équivalent, dont l'énoncé est le suivant :

> L'additionneur binaire avec retenue est un circuit effectuant la somme $S_i$ de 2 bits $A_i$ et $B_i$ d'ordre $i$ et d'une retenue $R_{i-1}$ d'ordre $i-1$ immédiatement inférieur. Effectuez la synthèse d'un tel circuit en utilisant, dans la mesure du possible, les opérateurs logiques *xor* ou *nxor*.

### 4.5.1 Nombre des entrées et sorties

Le circuit dispose de :
- 3 entrées : les 2 bits à additionner $A_i$ et $B_i$, et le bit de retenue de l'opération précédente $R_{i-1}$ ;
- 2 sorties : $S_i$ la somme des 2 bits et de la retenue précédente, et $R_i$ la nouvelle retenue générée par la somme.

### 4.5.2 Table de vérité du circuit

La table de vérité du circuit de l'additionneur est la suivante :

| $A_i$ $B_i$ $R_{i-1}$ | $S_i$ | $R_i$ |
|---|---|---|
| 0 0 0 | 0 | 0 |
| 0 0 1 | 1 | 0 |
| 0 1 0 | 1 | 0 |
| 0 1 1 | 0 | 1 |
| 1 0 0 | 1 | 0 |
| 1 0 1 | 0 | 1 |
| 1 1 0 | 0 | 1 |
| 1 1 1 | 1 | 1 |

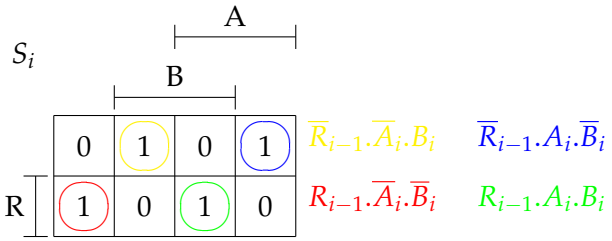
TAB. 4.3 – Table de vérité de l'additionneur avec retenue

### 4.5.3 Simplification par la méthode de Karnaugh

Nous allons par la suite utiliser la méthode de Karnaugh, séparément pour les deux sorties $S_i$ et $R_i$.

#### 4.5.3.1 La sortie $S_i$

Le tableau de Karnaugh de la sortie $S_i$ de l'additionneur est le suivant :




L'expression algébrique simplifiée déduite à partir du tableau de Karnaugh est :

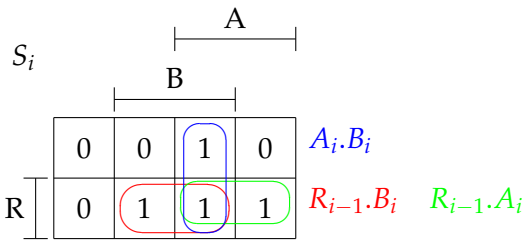
$$S_i = \overline{R}_{i-1}.\overline{A}_i.B_i + \overline{R}_{i-1}.A_i.\overline{B}_i + R_{i-1}.\overline{A}_i.\overline{B}_i + R_{i-1}.A_i.B_i$$

Dans l'énoncé, il est demandé d'utiliser les opérateurs logiques *xor* ou *nxor* pour avoir une simplification optimale. Nous continuons donc notre simplification en utilisant les lois et théorèmes de l'algèbre de Boole :

$$S_i = \overline{R}_{i-1}.(\overline{A}_i.B_i + A_i.\overline{B}_i) + R_{i-1}.(\overline{A}_i.\overline{B}_i + A_i.Bi)$$
$$S_i = \overline{R}_{i-1}.(A_i \oplus B_i) + R_{i-1}.\overline{(A_i \oplus B_i)}$$
$$S_i = R_{i-1} \oplus (A_i \oplus B_i)$$
$$S_i = R_{i-1} \oplus A_i \oplus B_i$$

#### 4.5.3.2 La sortie $R_i$

Le tableau de Karnaugh de la sortie $R_i$ de l'additionneur est le suivant :




L'expression algébrique simplifiée de la sortie $R_i$ déduite à partir du tableau de Karnaugh est :

$$R_i = R_{i-1}.A_i + R_{i-1}.B_i + A_i.B_i$$

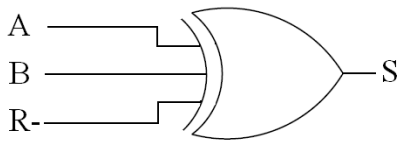
Dans l'énoncé, il est demandé d'utiliser les opérateurs logiques *xor* ou *nxor* pour avoir une simplification optimale. Nous continuons donc notre simplification en utilisant les lois et théorèmes de l'algèbre de Boole :

$$R_i = R_{i-1}.\overline{A}_i.B_i + R_{i-1}.A_i.\overline{B}_i + A_i.B_i$$
$$R_i = R_{i-1}.(\overline{A}_i.B_i + A_i.\overline{B}_i) + A_i.B_i$$
$$R_i = R_{i-1}.(A_i \oplus B_i) + A_i.B_i$$

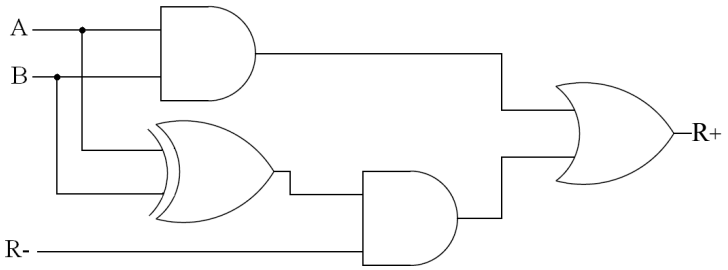
### 4.5.4 Le logigramme - schéma électronique

Après simplification des expressions algébriques des sorties, nous procédons à l'affichage du logigramme de chaque sortie :

– Pour la sortie $S_i$ :




– Pour la sortie $R_i$ :



## 4.6 Formalisation d'un schéma électronique

Pour pouvoir manipuler les schémas électroniques de façon formelle, nous avons besoin de définir le type abstrait « schéma électronique ». En effet, un type abstrait de données (TAD) (Guttag, 1975; Meyer, 2000) est une description mathématique formelle qui permet de représenter un type de données par ses propriétés et ses fonctionnalités, indépendamment de son implémentation physique en machine. Cette description est définie par un ensemble d'opérations, accompagnées optionnellement par des axiomes et des pré-conditions.

Une opération correspond à une fonctionnalité possible du TAD. Elle est identifiée par son nom, les ensembles de départ et les ensembles d'arrivée. Chaque opération est explicitée par une sémantique et est accompagnée d'un ou plusieurs axiomes que toutes les instances d'un TAD doivent vérifier à tout instant. Les pré-conditions définissent les domaines de validité des opérations.

Dans les sections suivantes, nous allons définir le type abstrait « schéma électronique » (SE), et certains des types abstraits nécessaires à sa construction. En effet, ce type utilise trois autres types abstraits :

- type abstrait de données « composant logique » : CL ;
- type abstrait de données « lien entre composants logiques » : Lien ;
- type abstrait de données « table de vérité » : TdV.

Grâce à cette représentation abstraite, nous parviendrons à formaliser les algorithmes de manipulation de schémas électroniques dans le reste du rapport, sans se soucier de leur implémentation physique.

### 4.6.1 Type abstrait « schéma électronique »

Nous présentons ci-dessous le type abstrait de données représentant les schémas électroniques (SE) à plusieurs entrées et une seule sortie :

| | | |
|---|---|---|
| **Nom**: | SE | |
| **Utilise**: | CL, Lien, TdV, **Naturel**, **Reel**, **Booleen**, Liste | |
| **Opérations**: | créerSE: | CL × Liste<SE> → SE |
| | estVideSE: | SE → **Booleen** |
| | obtenirNombreEntrées: | SE → **Naturel** |
| | obtenirNombreCLs: | SE → **Naturel** |
| | obtenirNombreLiens: | SE → **Naturel** |

| | | |
|---|---|---|
| | obtenirUneEntrée: | SE $\times$ **Naturel** $\nrightarrow$ CL |
| | obtenirSortie: | SE $\rightarrow$ CL |
| | obtenirListeCLs: | SE $\rightarrow$ Liste<CL> |
| | obtenirListeLiens: | SE $\rightarrow$ Liste<Lien> |
| | obtenirTdVSE: | SE $\rightarrow$ TdV |
| | obtenirRacineSE: | SE $\rightarrow$ CL |
| | obtenirNombreSousSEConnectés: | CL $\times$ SE $\rightarrow$ **Naturel** |
| | obtenirSousSEConnecté: | CL $\times$ SE $\times$ **Naturel** $\rightarrow$ SE |
| | obtenirListeSousSEConnectés: | CL $\times$ SE $\rightarrow$ Liste<SE> |
| **Sémantiques**: | créerSE: | retourne un SE ayant comme racine « CL » et comme sous-schémas connectés aux entrées de CL, ceux de « Liste ». |
| | estVideSE: | retourne **vrai** si un SE est vide, **faux** dans le cas contraire. |
| | obtenirNombreEntrées: | retourne le nombre des entrées du SE. |
| | obtenirNombreCLs: | retourne le nombre de composants logiques du SE. |
| | obtenirNombreLiens: | retourne le nombre de liens entre composants du SE. |
| | obtenirUneEntrée: | retourne l'entrée numéro $n$ du SE. |
| | obtenirSortie: | retourne la sortie du SE. |
| | obtenirListeCLs: | retourne la liste des composants logiques dans l'ordre croissant de leur identificateur respectif. |
| | obtenirListeLiens: | retourne la liste des liens entre les composants logiques. |
| | obtenirTdVSEE: | retourne la table de vérité du SE. |
| | obtenirRacineSE: | retourne le composant logique qui représente la racine du SE. |
| | obtenirNombreSousSEConnectés: | à partir d'un « CL », retourne le nombre de sous-schémas électroniques connectés à ce composant. |
| | obtenirSousSEConnecté: | retourne le sous-schéma électronique numéro $n$ connecté au composant logique « CL ». |
| | obtenirListeSousSEConnectés: | retourne la liste des sous-schémas électroniques connectés au composant logique |

« CL ».

**Préconditions**: obtenirUneEntrée(SE, n): $1 \leq n \leq$ obtenirNombreEntrées(SE)

### 4.6.2 Type abstrait « composant logique »

Pour les composants logiques, nous avons défini le type abstrait de données suivant :

**Nom**: CL

**Utilise**: SE, TypeComposant={*Entrée, Sortie, PorteLogique*}, TdV, **Naturel**, **Chaine**, **Booleen**, Liste

**Opérations**:

| | |
|---|---|
| créerCL: | **Chaine** × TypeComposant × **Naturel** → CL |
| typeCL: | CL → TypeComposant |
| étiquetteCL: | CL → **Chaine** |
| identificateurCL: | CL × SE → **Naturel** |
| obtenirArité: | CL → **Naturel** |
| obtenirTdVCL: | CL → TdV |

**Sémantiques**:

| | |
|---|---|
| créerCL: | retourne un nouveau composant logique. |
| typeCL: | retourne le type du composant logique - *Entrée, Sortie ou PorteLogique*. |
| étiquetteCL: | retourne l'étiquette du composant logigue, par exemple : *and*, *or*, $E_1$, $S_i$, etc. |
| identificateurCL: | retourne l'identificateur **unique** du composant logique par rapport à un SE. |
| obtenirArité: | retourne le nombre d'entrées du composant logique. Une entrée a une arité de 0 et une sortie a une arité de 1. |
| obtenirTdVCL: | retourne la table de vérité du composant logique. |

### 4.6.3 Type abstrait « Lien »

Pour les liens entre composants logiques, nous avons défini le type abstrait de données suivant :

| | | |
|---|---|---|
| **Nom**: | Lien | |
| **Utilise**: | CL | |
| **Opérations**: | créerLien: | CL × CL → Lien |
| | CLSource: | Lien → CL |
| | CLDestination: | Lien → CL |
| **Sémantiques**: | créerLien: | retourne un lien qui relie deux composants logiques. |
| | CLSource: | retourne le composant logique **source** du lien. Un composant logique de type *Sortie*, ne peut être source d'un Lien. |
| | CLDestination: | retourne le composant logique **destination** du lien. Un composant logique de type *Entrée* ne peut être destination d'un lien. |

## 4.7 Les schémas électroniques de la plate-forme TEB

### 4.7.1 Implémentation d'un schéma électronique dans la plate-forme TEB

Comme nous l'avons vu, la plate-forme TEB dispose du simulateur Logisim (Burch, 2002). Les schémas créés par ce simulateur existent sous un format XML particulier. Ce format reprend la « représentation graphique » du schéma électronique. Le diagramme de classes de la figure 4.12 illustre cette représentation graphique.

Pour les besoins de notre plate-forme, nous avons converti ce format « graphique » en un format « logique » plus facile à manipuler (Benomar, 2007).

En effet, le schéma électronique fourni par le simulateur Logisim est transformé en plusieurs schémas électroniques logiques, autant que le nombre de sorties du circuit d'origine. Chaque schéma logique obtenu dispose d'une seule sortie. Par exemple, un schéma électronique graphique ayant 3 entrées et 2 sorties, sera transformé en « deux »[9] schémas électroniques logiques ayant chacun 3 entrées et 1 sortie. Le diagramme de classes de la figure 4.13 illustre cette représentation logique.

[9] Un schéma pour chaque sortie du circuit d'origine.

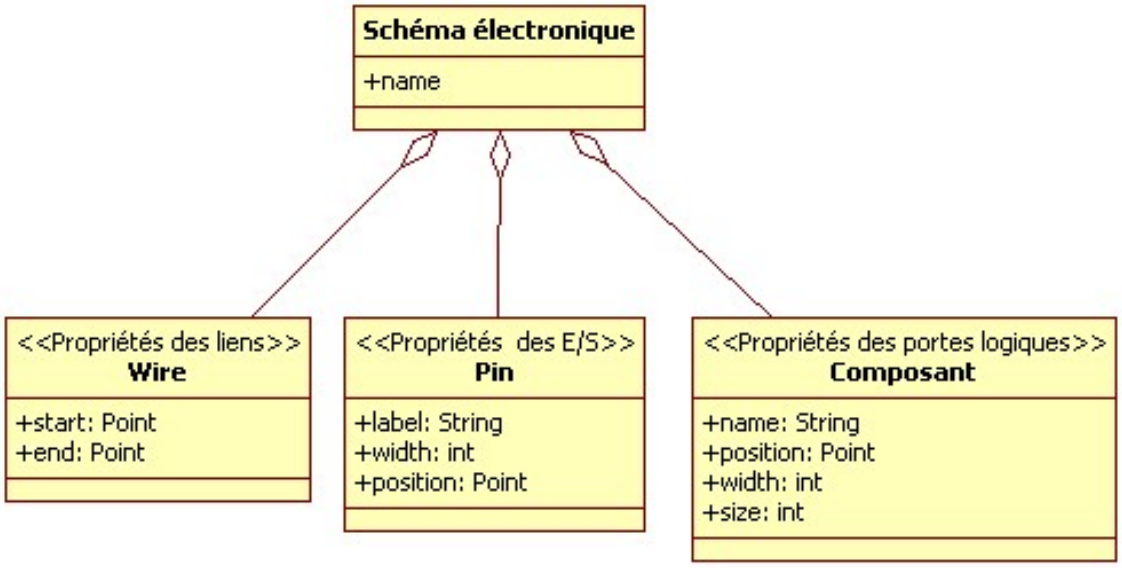



FIG. 4.12 – Diagramme de classes du format « graphique » d'un schéma électronique

Le format logique est représenté par un fichier XML dont la DTD (Document Type Definition)[10] est la suivante :

```
<!DOCTYPE Schéma [
<!ELEMENT Schéma(node+,edge+)>
<!ATTLIST Schéma Name CDATA \#REQUIRED>
<!ELEMENT node (\#PCDATA)>
<!ATTLIST node id ID \#REQUIRED>
<!ATTLIST node name CDATA \#REQUIRED>
<!ATTLIST node type CDATA \#REQUIRED>
<!ELEMENT edge (\#PCDATA)>
<!ATTLIST edge from CDATA \#REQUIRED>
<!ATTLIST edge to CDATA \#REQUIRED>
]>
```

Chaque balise et attribut de cette DTD ont une explication bien particulière :

- La balise « Schéma » : représente le schéma électronique logique. Elle est caractérisée par l'attribut « Name », qui est le nom de la sortie représentée par ce schéma ;
- La balise « node » : représente une porte logique du schéma électronique pour cette sortie. Elle a trois attributs : « id » (identificateur unique pour cette porte logique),

[10] La DTD est un modèle servant à définir un ensemble de règles pour la structure hiérarchique d'un document XML. C'est dans la DTD qu'on définit les différentes balises pouvant être utilisées (éléments ou nœuds), avec leurs attributs , auxquels on adjoint une définition.

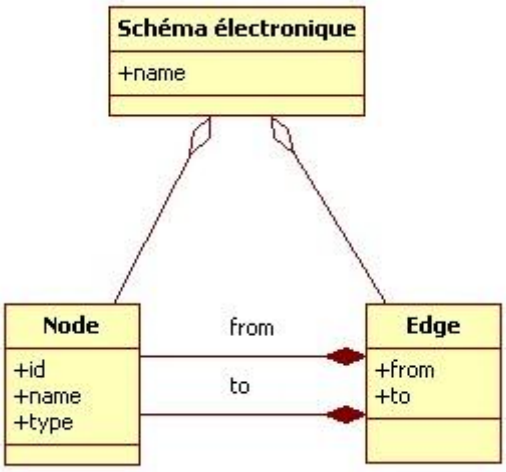



FIG. 4.13 – Diagramme de classes du format « logique » d'un schéma électronique

« name » (le nom de la porte logique : *not*, *and*, $A_i$, $S_i$, etc.) et « type » (le type de la porte logique : *In* pour *Entrée*, *Out* pour *Sortie* ou *Comp* pour *Composant*) ;
– La balise « edge » : représente les connexions entre les portes logiques. Elle a deux attributs : « from » (l'identificateur de la porte logique d'où part la connexion) et « to » (l'identificateur de la porte logique où arrive la connexion).

Une remarque importante concerne l'attribut « name » de la balise « node ». En effet, cet attribut est « unique », dans un même schéma électronique, pour les noms des portes logiques de type *In* ou *Out*, puisque par défaut, à la lecture d'un tel schéma, nous devons distinguer visuellement et rapidement les différentes entrées et sorties. Par contre, il est « générique » pour les portes logiques de type *Comp*, car il n'est pas important pour le format logique que nous utilisons, de distinguer les noms des composants logiques. Ceux-ci disposent déjà d'un identificateur unique qui est l'attribut « id ».

### 4.7.2 L'évaluation des circuits dans la plate-forme TEB

Actuellement, la plate-forme TEB ne permet pas d'effectuer la synthèse complète d'un circuit logique combinatoire, elle permet uniquement d'intégrer le schéma d'un circuit logique une fois la synthèse terminée. De cette manière, l'évaluation des schémas réalisés par les apprenants ne peut pas suivre la même démarche que celle dans le cas classique (cf. section 4.4 page 70).

Cette évaluation ne peut s'effectuer que sur les états de chaque sortie du circuit électronique et ne peut certifier avec exactitude à quelle étape, de la synthèse du circuit, cette erreur s'est produite. L'évaluation est donc uniquement « sommative » et nous informe uniquement sur l'état « correct » ou « inccorect » de la production de l'apprenant.

## 4.8 Conclusion

Nous souhaitons par la suite effectuer une évaluation « formative » à partir de la seule réalisation de l'apprenant : son schéma électronique. Or, ce résultat découle de toute une démarche intellectuelle qui a été décrite dans la section 4.4 page 70. Il peut donc y avoir des erreurs à n'importe quelle étape de la synthèse et notre objectif est donc de parvenir à repérer l'étape où l'erreur a été commise par l'apprenant et la caractériser.

La figure suivante propose un récapitulatif des étapes de la synthèse d'un circuit logique :

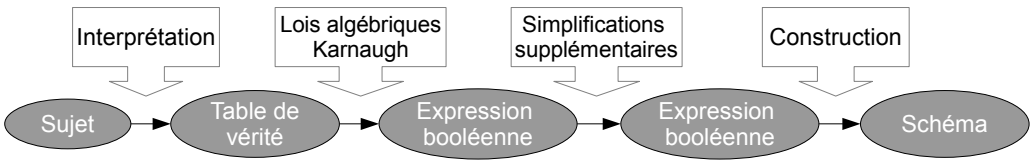



FIG. 4.14 – Les différentes étapes classiques de la synthèse d'un circuit logique

Nous pouvons regrouper ces erreurs en 3 types :

- Erreur de conception : dûe à une mauvaise interprétation initiale de l'énoncé (Ex : nombre d'entrées ou de sorties incorrect, mauvaise compréhension de l'énoncé), ce genre d'erreur provoque généralement une erreur dans la table de vérité ;
- Erreur de réalisation : dûe à la mauvaise utilisation des outils de synthèse (Ex : manque de connaissances quant à l'utilisation de la méthode de Karnaugh ou des simplifications algébriques) ;
- Erreur de construction : à ce stade, nous pouvons supposer que la table de vérité et les simplifications sont correctes, et que juste une confusion au niveau des symboles des portes logiques a provoqué cette erreur.

Notre hypothèse générale est que les algorithmes de classification vont nous permettre d'effectuer une « évaluation formative » des schémas électroniques produits par les apprenants en se basant uniquement sur la production finale (schéma électronique) et non sur le déroulement des différentes étapes de la synthèse. Pour la suite, avant de choisir un algorithme de classification, il nous faut construire une mesure de similarité entre nos données.

# Chapitre 5

# Proposition d'une mesure de similarité entre schémas électroniques

**Sommaire**

## Résumé

Notre domaine d'application étant l'électronique numérique, nous allons définir une mesure de similarité entre schémas électroniques de circuits logiques combinatoires.

Nous avons vu dans la section 4.2.4.3 page 66 qu'un circuit logique peut être représenté par un graphe orienté acyclique. En effet, les graphes sont utilisés comme modèle de représentation des connaissances. Ils peuvent ainsi modéliser des objets en utilisant leur représentation graphique naturelle, et il existe des techniques et outils mathématiques en théorie des graphes qui permettent de manipuler de tels objets. Nous allons donc nous baser sur les travaux existants sur les mesures de similarité entre graphes pour construire notre propre mesure de similarité.

Dans ce chapitre, nous commencerons par quelques définitions et notations. Nous présenterons quelques mesures de similarité existantes entre graphes. Nous définirons finalement notre propre mesure de similarité entre schémas électroniques que nous validerons à l'aide de données réelles.

## 5.1 Définitions et Notations

Dans cette section, nous allons commencer par quelques notations et définitions classiques sur les graphes que nous utiliserons dans la suite de ce rapport.

Un **graphe** $G = (V, E)$ est une structure formée d'un ensemble $V$ de $n$ nœuds ou sommets, et d'un ensemble $E \subseteq V \times V$ de $m$ arêtes reliant ces nœuds.

Un **graphe orienté** est un graphe dont les arêtes ont un sens bien précis, elles sont orientées. Dans ce cas, une arête est appelée un *arc*. Un arc reliant le sommet $u$ au sommet $v$ est noté $e = (u, v)$. Les extrémités de l'arc sont alors appelées extrémité initiale et extrémité finale.

Un **graphe acyclique orienté** est un graphe orienté qui ne possède pas de cycle ou boucle, c'est-à-dire, il ne permet le passage d'un sommet à un autre que dans un seul sens.

Soient les sommets $u, v \in V$, si $(u, v) \in E$ (respectivement $(v, u) \in E$) alors $(u, v)$ (respectivement $(v, u)$) est appelée **arête incidente** aux sommets $u$ et $v$ (respectivement aux sommets $v$ et $u$).

Un **sous-graphe** d'un graphe orienté ou non est le graphe obtenu en supprimant certains sommets et toutes les arêtes incidentes aux sommets supprimés. Par conséquent, $G' = (V', E')$ est un sous-graphe de $G = (V, E)$ si et seulement si $V' \subseteq V$ et $E' = E \cap (V' \times V')$.

Un **graphe complet** est un graphe où chaque sommet est relié à tous les autres.

Une **clique** d'un graphe $G = (V, E)$ est un sous-graphe complet de $G$.

Deux sommets $u$ et $v$ d'un graphe $G = (V, E)$ sont dit **adjacents**, s'ils sont reliés par une arête, c'est-à-dire $(u, v) \in E$ ou $(v, u) \in E$.

Un **graphe étiqueté** est défini par un tuple $G = (V, E, L_V, \alpha, L_E, \beta)$, où :

- $V$ est un ensemble de sommets ;
- $E \subseteq V \times V$ est l'ensemble des arcs entre ces sommets ;
- $L_V$ est un ensemble d'étiquettes de sommets ;
- $\alpha : V \rightarrow L_V$ est une application attribuant une étiquette à chaque sommet de $G$ ;
- $L_E$ est un ensemble d'étiquettes d'arcs ;
- $\beta : E \rightarrow L_E$ est une application attribuant une étiquette à chaque arc du graphe.

## 5.2 Mesure de similarité entre graphes

Les graphes sont souvent utilisés pour modéliser des objets dans divers domaines scientifiques et d'ingénierie tels que le traitement et reconnaissances d'images, la recherche d'information et les documents textuels, la représentation des données scientifiques, etc. (Conte *et al.*, 2004; Sorlin, 2006). Parmi ces applications, il existe celles qui représentent leurs objets en graphe afin de comparer leurs structures. Ceci correspond à résoudre un problème d'appariement entre graphes (ou *graph matching*) (Bunke et Messmer, 1997; Conte *et al.*, 2004).

Le principe d'appariement de graphes est de trouver une correspondance entre les sommets d'un graphe et les sommets d'un autre graphe, tout en préservant les liens entre ces sommets.

Il existe plusieurs catégories de problèmes d'appariement, nous définissons ci-après les quatre problèmes les plus connus (Bunke et Messmer, 1997; Sorlin, 2006) :

- l'isomorphisme de graphes, qui permet de vérifier que deux graphes sont structurellement identiques (Unger, 1964; Corneil et Gotlieb, 1970; Ullmann, 1976; Bunke et Messmer, 1997; Sorlin, 2006) ;
- l'isomorphisme de sous-graphes, qui permet de vérifier qu'un graphe est « inclus » dans un autre (Ullmann, 1976; Corneil et Gotlieb, 1970; Wong *et al.*, 1990; Bunke et Messmer, 1997; Sorlin, 2006) ;
- le plus grand sous-graphe commun, qui permet d'identifier la plus grande partie commune à deux graphes (Bunke, 1998; Messmer et Bunke, 1998; Sorlin, 2006) ;
- la distance d'édition de graphes (ou *Graph Edit Distance, GED*), qui permet d'identifier les opérations à effectuer pour transformer un graphe en un autre à un moindre coût (Tsai et Fu, 1979; Bunke et Allerman, 1983; Bunke, 1998; Sorlin, 2006).

Les deux premières catégories correspondent à un appariement « exact », où la méthode d'appariement requiert une correspondance stricte (isomorphisme de graphes) ou une inclusion stricte (isomorphisme de sous-graphes) entre les sommets et les arcs des deux graphes. Les deux dernières catégories correspondent à une méthode d'appariement « inexact », où la

correspondance entre les deux graphes revient à chercher le « meilleur » appariement possible, même si leurs structures sont différentes, moyennant quelques transformations.

Toutes ces catégories d'appariement nécessitent de définir formellement une mesure de similarité[1] entre deux graphes (Sorlin, 2006). Les mesures de similarité existantes dépendent du type d'appariement recherché. Par exemple, pour l'isomorphisme de graphes, la mesure de similarité consiste à définir l'existence d'une fonction d'isomorphisme entre deux graphes alors que la distance d'édition de graphes est basée sur la recherche d'un plus court chemin d'édition entre deux graphes (cf. section 5.2.4 page 87). D'un autre côté, leur classe de complexité est NP-complet ou NP-Difficile (Sorlin, 2006), c'est-à-dire que le temps de calcul augmente exponentiellement en fonction du nombre de sommets des graphes à comparer (Bunke et Messmer, 1997). Par conséquent, les algorithmes d'appariements entre graphes sont efficaces uniquement sur des graphes de petites tailles. Une taxonomie plus complète des différents algorithmes d'appariement est détaillée dans (Conte *et al.*, 2004).

### 5.2.1 Pourquoi mesurer la similarité entre deux graphes ?

Plusieurs raisons peuvent mener à mesurer la similarité entre graphes (Conte *et al.*, 2004; Sorlin, 2006), nous pouvons citer entre autres :

- Reconnaissance de formes et vision par ordinateur : reconnaître des images 2D ou 3D, des séquences vidéo, des caractères, des symboles, des graphiques, etc ;
- Recherche d'information : trouver des documents similaire à une requête, etc ;
- Classification et « *machine learning* » : regrouper des objets similaires ;
- Raisonnement à partir de cas : trouver un problème déjà résolu similaire au problème à résoudre ;
- Conception logicielle : retrouver des occurrences d'un code à l'intérieur d'un autre code.

Plusieurs applications de classification utilisant la mesure de similarité existent, nous pouvons citer (Neuhaus et Bunke, 2005) pour la classification des empreintes digitales, (Pearce *et al.*, 1994) pour la reconnaissance des formes, (Jiang *et al.*, 2000) pour la reconnaissance des symboles et (Saux et Bunke, 2000) pour la classification des images.

Un autre exemple est celui de D. Genest, il s'est intéressé à l'utilisation de graphes particuliers pour indexer des ressources bibliographiques (Genest, 2000). Il a développé un système qui permet à un utilisateur de proposer une requête sous la forme d'un graphe. Cette requête est alors analysée, et comparée à chacun des graphes ayant servi à indexer les documents de la base. Le système effectue des transformations plus ou moins importantes du graphe requête afin d'identifier les documents qui sont proches de la requête. De cette manière, il est possible de classer les réponses par ordre de pertinence en fonction du nombre de transformations effectuées.

[1] Nous pouvons aussi dire « dissimilarité ».

Dans le cas de la reconnaissance des formes, l'application principale est l'indexation d'images. Pour déterminer la classe d'appartenance d'un objet, il est comparé à d'autres objets d'une base d'apprentissage à l'aide de fonctions de similarité. C'est dans ce cadre que F. Suard (2006) a proposé une méthode de détection de piéton en utilisant des graphes étiquetés pour modéliser le corps humain. L'utilisation des étiquettes permet de résoudre les problèmes de variabilités d'apparence, de taille et de posture. Le graphe est construit à partir du squelette morphologique du masque binaire du corps humain. La méthode de classification utilisée fait appel à des machines à noyaux, en l'occurrence le classifieur SVM (Support Vector Machine).

### 5.2.2 Comment mesurer la similarité entre deux graphes ?

Toutes les applications référencées dans la section précédente nécessitent de définir formellement une mesure de similarité afin de comparer leurs objets qui sont représentés sous format de graphes :

> *« Lorsque les objets sont modélisés par les graphes, il est donc nécessaire de définir une mesure de similarité (ou de distance) entre tout couple de graphes. Comparer deux graphes consiste à comparer leur structure, c'est-à-dire la répartition de leurs arcs sur leurs sommets. Les sommets des deux graphes sont mis en correspondance et la similarité des structures dépend du nombre d'arcs que cette mise en correspondance des sommets permet de "retrouver".*
>
> *Nous avons vu que dans de nombreuses applications les sommets et les arcs des graphes sont valués afin de qualifier les différents composants et les différentes relations entre ces composants. Par conséquent, dans ces applications, la distance entre deux graphes dépendra de leur structure mais également des différences entre les valeurs portées par les sommets et les arcs mis en correspondance. »* (Sorlin, 2006)

Par la suite, nous allons nous intéresser aux mesures qui vont nous permettre de « quantifier » la similarité ou la dissimilarité entre les structures de deux graphes, c'est-à-dire les mesures de similarité qui permettent « d'évaluer » la ressemblance entre deux graphes dans le cas des appariements inexacts. Ce qui correspond aux techniques de comparaison de graphes à tolérance d'erreurs telles que la recherche du plus grand sous-graphe commun ou la distance d'édition de graphes.

### 5.2.3 Plus grand sous-graphe commun

Le principe est le suivant : $g_3$ est un plus grand sous-graphe commun à deux graphes $g_1$ et $g_2$ (noté $mcs(g_1, g_2)$ pour *maximum common subgraph*), si $g_3$ est un sous-graphe commun à $g_1$ et $g_2$ ayant un nombre de sommets maximum, c'est-à-dire qu'il n'existe aucun autre sous-

graphe commun à $g_1$ et $g_2$ avec un ordre (ou nombre de sommets) strictement plus grand que celui de $g_3$ (voir exemple de la figure 5.1).

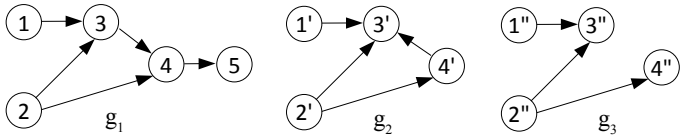



FIG. 5.1 – Exemple de sous-graphe commun. $g_3$ est un plus grand sous-graphe commun à $g_1$ et $g_2$.

Dans (Bunke et Shearer, 1998; Bunke, 2000), BUNKE propose de définir une mesure de similarité entre deux graphes basée sur la méthode du plus grand sous-graphe commun par :

$$\boxed{d(g_1, g_2) = 1 - \frac{|mcs(g_1, g_2)|}{max(|g_1|, |g_2|)}} \tag{5.1}$$

où $|mcs(g_1, g_2)|$ est le nombre de sommets du plus grand sous-graphe commun à $g_1$ et $g_2$, $|g_1|$ (respectivement $|g_2|$) est le nombre de sommets du graphe $g_1$ (respectivement $g_2$) et $0 \leq d(g_1, g_2) \leq 1$.

Ce que nous pouvons constater à partir de cette équation est que, plus le nombre d'ordre du « plus grand sous-graphe commun » est grand, plus le rapport $\frac{|mcs(g_1, g_2)|}{max(|g_1|, |g_2|)}$ est proche de 1, et donc $d(g_1, g_2)$ est proche de 0. Ce qui signifie que meilleure est la similarité entre les deux graphes.

Toujours selon BUNKE, cette mesure est une *distance métrique* et respecte donc les conditions suivantes :

- $d(g_1, g_2) = 0 \leftrightarrow g_1$ et $g_2$ sont isomorphes [2].
- $d(g_1, g_2) = d(g_2, g_1)$
- $d(g_1, g_3) \leq d(g_1, g_2) + d(g_2, g_3)$

Cette mesure de similarité nécessite préalablement la recherche du plus grand sous-graphe commun (*mcs*). Les algorithmes classiques pour calculer le *mcs* de deux graphes sont basés soit sur la détection de clique de taille maximale (Levi, 1972), soit sur une exploration exhaustive d'un arbre de recherche avec « backtracking » (retour-arrière) (McGregor, 1982). D'autres algorithmes sont aussi présentés dans les travaux de recherches de B. CUISSART (2004).

Par contre, cette mesure consiste uniquement à déterminer les points communs des deux graphes en faisant abstraction de leurs différences. Elle utilise un calcul de distance qui tient compte seulement de ces ressemblances. C'est donc une méthode « quantitative » et non « qualitative », dont la mesure qu'elle ne permet pas de détecter les différences entre les deux graphes afin de les corriger éventuellement.

[2] Les deux graphes ont une structure identique.

Dans la suite, nous allons nous intéresser à une autre méthode d'appariement de graphes qui permet, en plus du calcul de leur mesure de similarité, de détecter les différences entre ses graphes.

### 5.2.4 Distance d'édition de graphes

Une alternative à la mesure de similarité construite à partir du « plus grand sous-graphe commun » est la distance d'édition de graphe (Bunke, 1997). Cette méthode est très largement utilisée pour déterminer des appariements de graphes tolérants aux erreurs. Dans (Bunke et Allerman, 1983; Messmer et Bunke, 1998; Neuhaus *et al.*, 2006; Riesen et Bunke, 2008), BUNKE a proposé plusieurs algorithmes de mesure de similarité basés sur la distance d'édition de graphes ou « Graph Edit Distance (GED) ». Le principe de ces algorithmes est le suivant : étant donné deux graphes $g_1$ et $g_2$, l'idée est d'appliquer une séquence de transformations (ou d'opérations d'édition) sur $g_1$ (insertion, substitution et suppression de sommets ou d'arcs) pour finalement transformer $g_1$ en $g_2$. Ce processus tend à déformer le modèle du graphe initial jusqu'à ce qu'il s'identifie avec le modèle de l'autre graphe.

Chaque transformation ayant un coût prédéfini, le coût d'une séquence de transformations est égale à la somme pondérée des coûts de chaque transformation. La séquence de transformations qui obtient un coût minimal représente la distance d'édition entre les deux graphes (Bunke, 1997) :

$$\boxed{d(g_1, g_2) = \min_{t \in E(g_1, g_2)} \{ \sum_{i=1}^{k=card(t)} c(e_i) \}} \tag{5.2}$$

« $e_i$ » est appelé une opération d'édition. « $t = (e_1, ....e_k)$ » est une séquence d'opérations d'édition transformant $g_1$ en $g_2$, aussi appelé « chemin d'édition complet ».

Un « **chemin d'édition complet** » est une séquence de transformations qui permettent de transformer complètement $g_1$ en $g_2$. D'un autre côté, un « **chemin d'édition partiel** » est une séquence de transformations qui permettent de transformer partiellement $g_1$ afin de s'approcher de la forme de $g_2$.

$E(g_1, g_2)$ est l'ensemble des séquences d'opérations d'édition qui permettent de transformer $g_1$ en $g_2$. « $c$ » est la fonction qui assigne un coût à une opération d'édition « $e$ ».

#### 5.2.4.1 Opérations d'édition

Comme nous venons de le voir, une opération d'édition transforme partiellement un graphe. Une transformation est effectuée soit sur un sommet, soit sur un arc. À chaque transformation est associé un coût (Bunke et Allerman, 1983).

Il existe six types de transformations possibles :

- *Substitution de sommet* : cette opération correspond au remplacement d'un sommet par un autre. Le coût de cette opération correspond au coût de remplacement entre les deux étiquettes des deux sommets ;
- *Suppression de sommet* : cette opération correspond à la suppression d'un sommet et de tous les arcs le reliant aux sommets adjacents. Le coût totale de cette opération est égale au coût de suppression d'un sommet, plus les coûts de suppression de tous les arcs incidents à ce sommet ;
- *Insertion de sommet* : cette opération correspond à l'ajout d'un sommet et éventuellement les arcs le reliant aux autres sommets. Le coût totale de cette opération est égale au coût d'insertion d'un sommet, plus les coûts d'insertion de tous les arcs incidents insérés ;
- *Substitution d'arc* : cette opération correspond à remplacer un arc par un autre. Le coût de cette opération correspond au coût de remplacement entre les deux étiquettes des deux arcs ;
- *Suppression d'arc* : cette opération correspond à la suppression d'un arc reliant deux sommets. Le coût de cette opération est égale au coût de suppression d'un arc ;
- *Insertion d'arc* : cette opération correspond à l'ajout d'un arc entre deux sommets. Le coût de cette opération est égale au coût d'insertion d'un arc.

Suivant les valeurs affectées à chaque type de coût (coûts d'insertion, de substitution ou de suppression), la distance d'édition de graphes est une *distance métrique* (Bunke et Allerman, 1983). Soit $d$ une distance d'édition de graphes. $g_1 = (V_1, E_1)$ et $g_2 = (V_2, E_2)$ deux graphes étiquetés non vide. $d$ est une métrique, si les conditions suivantes sont remplies :

$\forall\ n \in V_1$ et $v \in V_2$ ; $\forall\ i \in E_1$ et $j \in E_2$

(1) $c_{ns}(n, v) = 0$ si les deux sommets $n$ et $v$ ont la même étiquette, $c_{ns}(n, v) > 0$ sinon, et $c_{es}(i, j) = 0$ si les deux arcs $i$ et $j$ ont la même étiquette, $c_{es}(i, j) > 0$ sinon ;

(2) $c_{nd}(n) > 0$, $c_{ni}(n) > 0$ et $c_{ed}(i) > 0$, $c_{ei}(i) > 0$ ;

(3) $c_{ns}(n, v) = c_{ns}(v, n)$, $c_{nd}(n) = c_{ni}(n)$ et $c_{es}(i, j) = c_{es}(j, i)$, $c_{ed}(i) = c_{ei}(j)$.

où,

- $c_{ns}(n, v)$ est le coût de substitution du sommet $n$ par le sommet $v$ ;
- $c_{nd}(n)$ est le coût de suppression du sommet $n$ ;
- $c_{ni}(n)$ est le coût d'insertion du sommet $n$ ;
- $c_{es}(i, j)$ est le coût de substitution de l'arc $i$ par l'arc $j$ ;
- $c_{ed}(i)$ est le coût de suppression de l'arc $i$ ;
- $c_{ei}(i)$ est le coût d'insertion de l'arc $i$.

#### 5.2.4.2 Algorithme de calcul de la distance d'édition de graphes

Généralement, les algorithmes de « distance d'édition de graphes » utilisent une méthode de recherche basée sur A*[3] (Hart *et al.*, 1968; Nilsson, 1980; Bunke et Allerman, 1983). L'idée est de représenter le problème d'appariement par un arbre de recherche construit dynamiquement de façon itérative, dont la racine est l'état initial, les nœuds intermédiaires représentent des solutions partielles et les feuilles sont des solutions complètes.

Ce sont des algorithmes admissibles, en ce sens qu'ils garantissent de trouver un « appariement inexact optimal » entre les deux graphes (Nilsson, 1980). Cependant, de part leur complexité, ils ne peuvent fonctionner que sur des graphes relativement petits. Par contre, afin de réduire le temps et la complexité d'exécution, ces algorithmes utilisent diverses techniques heuristiques qui sont souvent dépendantes du domaine d'application.

Nous utiliserons par la suite, l'algorithme de distance d'édition de graphes comme il est défini dans (Neuhaus *et al.*, 2006; Riesen et Bunke, 2008). Cet algorithme (cf. Algorithme 5), nous garantit l'obtention d'un *chemin d'édition complet ayant un coût minimal*. La substitution d'un sommet est notée : $u \rightarrow v$, l'insertion d'un sommet est notée : $\varepsilon \rightarrow u$ et la suppression $u \rightarrow \varepsilon$.

Cet algorithme explore toutes les possibilités de correspondances entre les sommets et les arcs du graphe $g_1$ par rapport aux sommets et arcs du graphes $g_2$ grâce à l'utilisation de l'algorithme de recherche A*. $AR$[4] contient la construction dynamique de l'arbre de recherche A* dont les nœuds $p$ représentent les séquences de transformations partielles[5], et les feuilles, les séquences de transformations complètes[6]. Pour chaque séquence de transformations $p$ de l'arbre de recherche, $g(p)$ est le coût des transformations effectuées, et $h(p)$ est le « coût estimé » des transformations restantes jusqu'à la solution finale. Dans le cas où aucune heuristique n'est utilisée, la valeur de $h(p)$ est mise à zéro.

Initialement, $AR$ est vide (ligne 1). Ensuite, toutes les opérations possibles de substitution et suppression sont générées simultanément dans l'arbre de recherche $AR$ pour chaque sommet (lignes 2-3 et 9-10), et si tous les sommets du premier graphe ont été traités, le reste des sommets du deuxième graphe sont insérés dans $AR$ en une seule opération (ligne 12).

Les opérations d'édition sur les arcs sont implicites par les opérations d'édition sur leurs sommets adjacents, c'est-à-dire, qu'un arc est substitué, supprimé ou inséré en fonction de l'opération d'édition effectuée sur ses sommets adjacents. Soit par exemple les graphes $g_1 = (V_1, E_1)$ et $g_2 = (V_2, E_2)$, et soit $u, u' \in V_1 \cup \{\varepsilon\}$ et $v, v' \in V_2 \cup \{\varepsilon\}$. Supposons que les

[3] Un algorithme qui permet de trouver le meilleur chemin à travers le parcours en profondeur d'un arbre de recherche en utilisant des heuristiques.

[4] Arbre de Recherche.

[5] Ensemble de transformations entre certains sommets et arcs des graphes $g_1$ et $g_2$ qui nous rapprochent de la solution finale.

[6] Ensemble de transformations entre les sommets et arcs des graphes $g_1$ et $g_2$ correspondant à une solution finale.

**Algorithme 5** Distance d'édition de graphes basée sur l'algorithme A*

**Données:**

$g_1 = (V_1, E_1, \alpha_1, \beta_1)$ et $g_2 = (V_2, E_2, \alpha_2, \beta_2)$ deux graphes non vide,
avec $V_1 = \{u_1, ..., u_{|V_1|}\}$ et $V_2 = \{v_1, ..., v_{|V_2|}\}$

**Résultats:**

Un chemin d'édition complet entre $g_1$ et $g_2$ ayant un coût minimal
c'est-à-dire $p_{min} = \{u_1 \rightarrow v_3, u_2 \rightarrow \varepsilon, ..., \varepsilon \rightarrow v_6\}$

$AR \leftarrow \emptyset$
**pour chaque** sommet $w \in V_2$, insérer la substitution $\{u_1 \rightarrow w\}$ dans $AR$
**insérer** la suppression $\{u_1 \rightarrow \varepsilon\}$ dans $AR$
**répéter**
  **enlever** $p_{min} \leftarrow argmin_{p \in AR}\{g(p) + h(p)\}$ de $AR$
  **si** $p_{min}$ n'est pas un chemin d'édition complet **alors**
    soit $p_{min} = \{u_1 \rightarrow v_{i_1}, ..., u_k \rightarrow v_{i_k}\}$
    **si** $k < |V_1|$ **alors**
      **pour chaque** $w \in V_2 \backslash \{v_{i_1}, ..., v_{i_k}\}$, insérer $p_{min} \cup \{u_{k+1} \rightarrow w\}$ dans $AR$
      **insérer** $p_{min} \cup \{u_{k+1} \rightarrow \varepsilon\}$ dans $AR$
    **sinon**
      **insérer** $p_{min} \cup \bigcup_{w \in V_2 \backslash \{v_{i_1}, ..., v_{i_k}\}} \{\varepsilon \rightarrow w\}$ dans $AR$
    **fin si**
  **fin si**
**jusqu'à ce que** $p_{min}$ soit un chemin d'édition complet
**retourner** $p_{min}$

opérations d'édition suivantes $(u \rightarrow v)$ et $(u' \rightarrow v')$ ont été exécutées. Nous distinguons trois cas possibles (Riesen et Bunke, 2008) :

(1) si $e_1 = (u, u') \in E_1$ et $e_2 = (v, v') \in E_2$, alors l'opération de substitution d'un arc $(e_1 \rightarrow e_2)$ est implicite par les opérations d'édition ci-dessus ;

(2) si $e_1 = (u, u') \in E_1$ et $e_2 = (v, v') \notin E_2$, alors l'opération de suppression d'un arc $(e_1 \rightarrow \varepsilon)$ est implicite par les opérations d'édition ci-dessus ;

(3) si $e_1 = (u, u') \notin E_1$ et $e_2 = (v, v') \in E_2$, alors l'opération d'insertion d'un arc $(\varepsilon \rightarrow e_2)$ est implicite par les opérations d'édition ci-dessus.

Finalement, à chaque itération de l'algorithme, un chemin partial $p$ ayant le coût minimum est choisi à partir de l'arbre de recherche (ligne 5). Si $p$ correspond à un chemin d'édition complet (ligne 15), il correspond au chemin d'édition optimal dont le coût est la distance d'édition entre les deux graphes. D'un autre côté, dans le cas où le temps de calcul dépasse un seuil prédéfini, la distance d'édition de graphes correspondante est considérée comme infinie (Neuhaus *et al.*, 2006). Ce dernier cas n'est pas représenté dans l'algorithme.

#### 5.2.4.3 Exemple de calcul de la distance d'édition de graphes

Nous allons appliquer l'algorithme de la distance d'édition de graphes sur un exemple simple afin de comprendre son fonctionnement. Prenons les deux graphes de la figure 5.2, avec les coûts suivants pour chaque type d'opération :

- $c_{ns}(n, v) = 1$ si les étiquettes des sommets $n$ et $v$ sont différentes, $c_{ns}(n, v) = 0$ si les étiquettes des sommets $n$ et $v$ sont identiques ;
- $c_{nd}(n) = c_{ni}(n) = 2$, les coûts de suppression et d'insertion du sommet $n$ ;
- $c_{es}(i, j) = 1$ si les étiquettes des arcs $i$ et $j$ sont différentes, $c_{es}(i, j) = 0$ si les étiquettes des arcs $i$ et $j$ sont identiques ;
- $c_{ed}(i) = c_{ei}(i) = 1$, les coût de suppression et d'insertion de l'arc $i$.

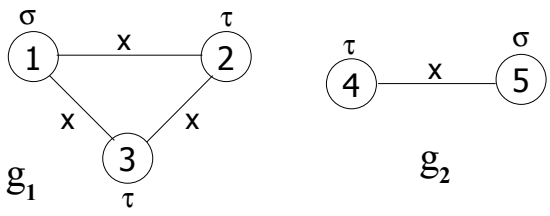



FIG. 5.2 – Exemple de deux graphes (Bunke et Allerman, 1983).

La figure 5.3 représente l'arbre de recherche qui permettra de trouver un appariement inexact optimal entre les graphes $g_1$ et $g_2$. L'état initial étant l'ensemble vide, les états intermédiaires représentent les chemins d'édition partiels et les états finaux (les feuilles de l'arbre) sont les chemins d'édition complets. Les coûts des transformations sont affichés à l'intérieur

des cercles. L'un des chemins d'édition complets qui obtient le coût minimal est la distance d'édition entre les deux graphes.

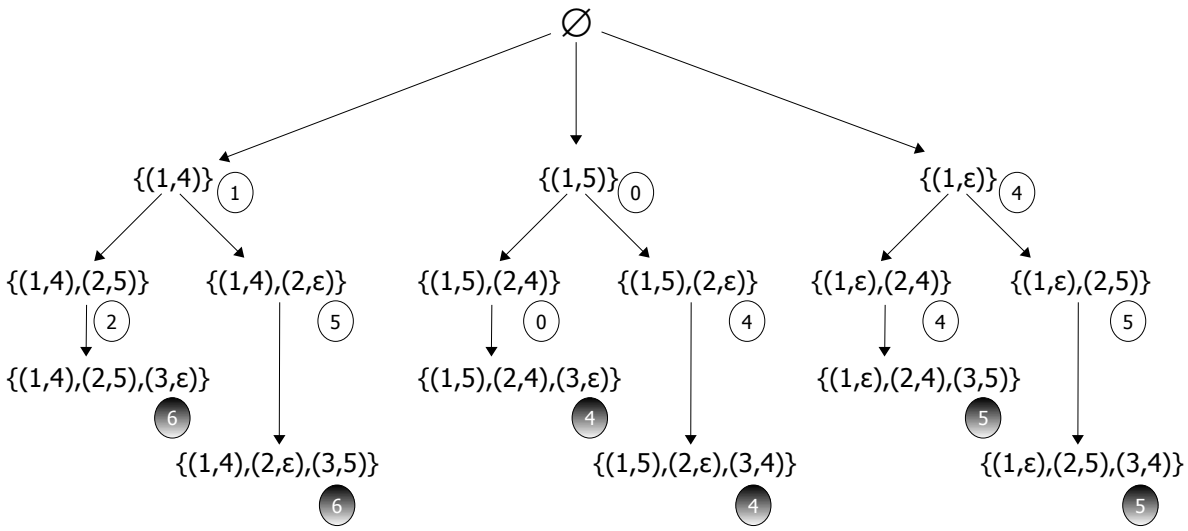



FIG. 5.3 – L'espace de recherche pour l'appariement des deux graphes $g_1$ et $g_2$.

D'après NILSSON (1980), cette méthode de recherche est admissible dans la mesure qu'elle garantit de toujours trouver un chemin d'édition complet avec un coût minimal.

## 5.3 Mesure de similarité entre schémas électroniques

### 5.3.1 Particularité des schémas électroniques

#### 5.3.1.1 Un schéma électronique comme un ensemble de graphes

Nous avons vu dans le chapitre 4 que les schémas électroniques pouvaient être représentés par des graphes orientés acycliques, avec en plus la particularité que les sommets sont « étiquetés » (les étiquettes sont les noms des portes logiques) et les arcs sont « non étiquettés », mais ont une direction (toujours dans le sens des entrées vers les sorties). Prenons l'exemple du schéma de la figure 5.4. Il représente le circuit de l'additionneur binaire avec retenue. Il est composé de 3 entrées ($A_i$ et $B_i$ les bits à additionner, et $R_{i-1}$ la retenue de l'addition précédente) et de 2 sorties ($S_i$ l'addition binaire et $R_i$ la nouvelle retenue générée par cette opération).

Ce schéma électronique peut être représenté par le graphe de la figure 5.5, où les portes logiques (ou composants) sont les sommets du graphe avec comme étiquette le nom de la porte, et les connexions entre les portes logiques sont les arcs du graphe. De plus, ce graphe est orienté acyclique car, quand nous partons d'une des entrées et que nous nous déplaçons

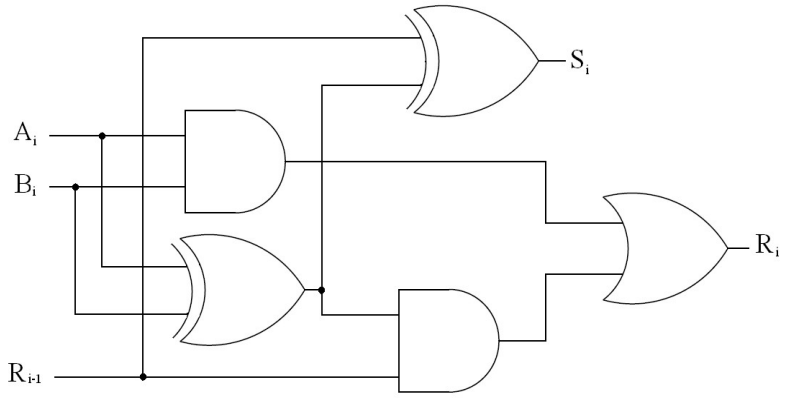



FIG. 5.4 – Exemple du schéma électronique de l'additionneur complet avec retenue

de porte en porte, en suivant les connexions, nous atteignons nécessairement une des sorties du circuit. Ces deux représentations sont donc identiques, et nous utiliserons par la suite l'une ou l'autre pour illustrer un schéma électronique.

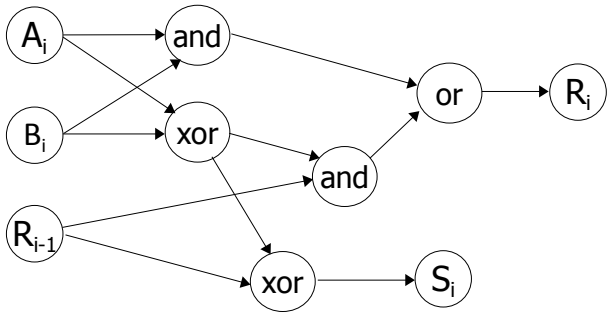



FIG. 5.5 – Exemple du graphe-circuit de l'additionneur complet avec retenue

Dans le chapitre 4, nous avons aussi détaillé les étapes classiques de la synthèse d'un circuit logique. Parmi ces étapes, il y a la simplification de la fonction logique de chaque sortie du circuit logique. Pour cette raison, nous sommes amenés à étudier séparément chaque sortie du circuit, comme si elle correspondait à un circuit à part entière avec une seule sortie. La figure 5.6 affiche les deux graphes-circuits représentant séparément les deux sorties $S_i$ et $R_i$ de l'additionneur binaire avec retenue.

Cette particularité des circuits électroniques, nous permet de décomposer un schéma en plusieurs graphes-circuits ayant un seul sommet particulier appelé « racine », d'où aucun arc n'est sortant. Ce sommet correspond généralement à une des sorties du circuit logique. Ces graphes-circuits disposent aussi d'autres sommets particuliers, appelées « feuilles », d'où aucun arc n'est entrant. Ces sommets correspondent aux entrées du circuit logique. Cette décomposition, en plusieurs graphes-circuits, nous permettra ensuite d'utiliser les algorithmes de comparaisons de schémas électroniques de façon plus simple.

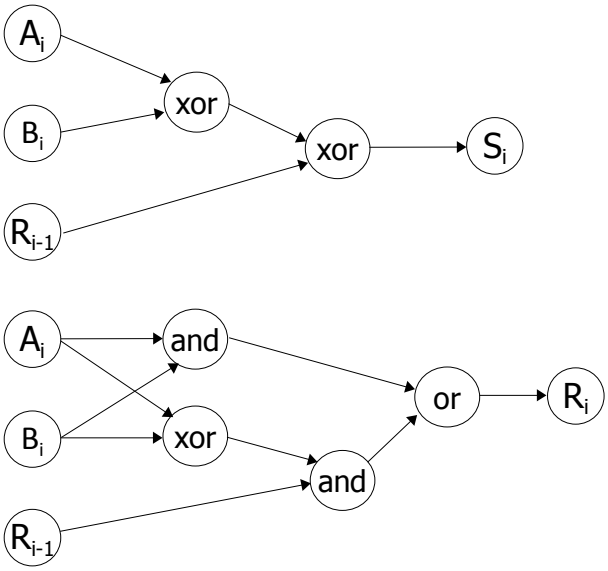



FIG. 5.6 – Les graphes-circuits des 2 sorties de l'additionneur binaire

#### 5.3.1.2 Schémas électroniques équivalents

La notion d'équivalence entre schémas électroniques se base sur les théorèmes et lois de l'algèbre booléenne (Darche, 2002; Zanella et Ligier, 1998). En effet, les deux lois représentées par les opérateurs *and* et *or* sont commutatives et associatives, et la transformation d'une expression algébrique en un schéma électronique peut avoir plusieurs résultats graphiquement différents, mais équivalents vis à vis de la fonction logique correspondante. Ceci est généralement dû à l'arité variable de certains composants électroniques.

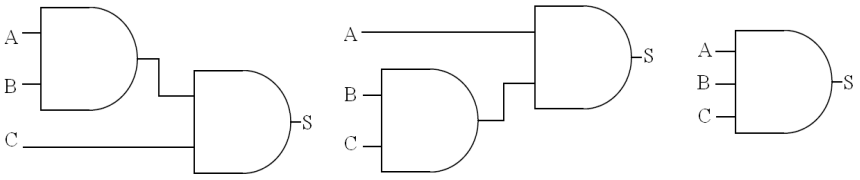



FIG. 5.7 – Exemple de schémas électroniques équivalents.

Par exemple, les schémas électroniques de la figure 5.7 correspondent à la même fonction logique ($S = A.B.C$), et sont donc « équivalents », même si leurs représentations graphiques sont différentes. Cette différence est due à l'utilisation de portes logiques d'arité 2 dans les deux premiers schémas et de porte logique d'arité 3 dans le dernier schéma.

La distance d'édition de graphes entre ces schémas, considérés comme des graphes, telle qu'elle a été définie par l'algorithme de BUNKE et al. (cf. 5 page 90), n'est pas égale à zéro. Du point de vue structurel ces schémas ne sont donc pas identiques. Ceci pourrait « fausser » le

résultat des mesures de similarité entre schémas électroniques si nous ne tenions pas compte de cette particularité d'équivalence.

Pour cela, nous avons introduit la notion d'équivalence entre schémas électroniques dans la construction de notre mesure de similarité afin de tenir compte de l'arité variable des composants électroniques. Les algorithmes qui permettent de formaliser cette notion sont présentés dans les paragraphes suivants. Ces algorithmes utilisent le type abstrait de données « schéma électronique », introduit dans le chapitre 4.

**Algorithme 6** SCHÉMAS ÉLECTRONIQUES ÉQUIVALENTS

**Données:**

$SE_1$, $SE_2$ deux schémas électroniques non vide,

**Résultats:**

= **vrai**, si les deux schémas sont équivalents,

= **faux**, dans le cas contraire.

1: $SE_1 \leftarrow$ MAXIMISER_ARITÉS($SE_1$)
2: $SE_2 \leftarrow$ MAXIMISER_ARITÉS($SE_2$)
3:
4: **retourner** SCHÉMAS_ELECTRONIQUES_IDENTIQUES($SE_1$,$SE_2$)

L'algorithme d'équivalence entre schémas électroniques nécessite la formalisation de deux autres algorithmes qui sont :

1. MAXIMISER_ARITÉ : qui permet de tenir compte de l'arité des portes logiques pour obtenir un schéma électronique qui maximise les arités de toutes les portes logiques du schéma d'origine, tout en gardant les mêmes résultats pour la table de vérité correspondante ;
2. SCHÉMAS_ELECTRONIQUES_IDENTIQUES : qui tient compte de la loi algébrique de commutativité des portes logiques pour finalement vérifier que les deux schémas électroniques en entrée sont identiques à quelques permutations prêt des entrées des portes logiques.

**Algorithme 7** MAXIMISER ARITÉS

**Données:**

$SE$ un schéma électronique non vide,

**Résultats:**

le schéma électronique dont l'arité de toutes ses portes logiques a été maximisée.

```
1:  R ← obtenirRacineSE(SE)
2:  si typeCL(R)=Entrée alors
3:    retourner SE
4:  sinon
5:    arité_SE_modifiée ← faux
6:    nouveaux_SSE ← () // Liste vide de schémas électroniques
7:    pour chaque SSE dans obtenirListeSousSEConnectés(R, SE) faire
8:      SSE_temp ← MAXIMISER_ARITÉ(SSE)
9:      R_temp ← obtenirRacineSE(SSE_temp)
10:     si typeCL(R_temp) = PorteLogique alors
11:       si EtiquetteCL(R) = EtiquetteCL(R_temp) alors
12:         nouveaux_SSE ← nouveaux_SSE + obtenirListeSousSEConnectés(R_temp, SSE_temp)
13:         arité_SE_modifiée ← vrai
14:       sinon
15:         nouveaux_SSE ← nouveaux_SSE + SSE_temp
16:       fin si
17:     sinon
18:       nouveaux_SSE ← nouveaux_SSE + SSE_temp
19:     fin si
20:   fin pour
21:   si arité_SE_modifiée alors
22:     retourner MAXIMISER_ARITÉ(créerSE(R, nouveaux_SSE))
23:   sinon
24:     retourner créerSE(R, nouveaux_SSE)
25:   fin si
26: fin si
```

**Algorithme 8** Schémas Électroniques Identiques

**Données:**

$SE_1$, $SE_2$ deux schémas électroniques non vide,

**Résultats:**

= **vrai**, si les deux schémas sont identiques,

= **faux**, dans le cas contraire.

```
1: R1 ← obtenirRacineSE(SE1)
2: R2 ← obtenirRacineSE(SE2)
3: n1 ← obtenirNombreSousSEConnectés(R1, SE1)
4: n2 ← obtenirNombreSousSEConnectés(R2, SE2)
5: si (EtiquetteCL(R1)≠ EtiquetteCL(R2)) ou (n1 ≠ n2) alors
6:   retourner faux
7: sinon
8:   sont_identiques ← vrai
9:   ListeSSE1 ← obtenirListeSousSEConnectés(R1, SE1)
10:  ListeSSE2 ← obtenirListeSousSEConnectés(R2, SE2)
11:  tant que sont_identiques et non estVide(ListeSSE1) faire
12:    SSE1 ← obtenirElement(ListeSSE1,1)
13:    supprimer(ListeSSE1,1)
14:    ListeSSE2_temp ← ListeSSE2
15:    a_trouvé_SE_associé ← faux
16:    tant que (non a_trouvé_SE_associé) et (non estVide(ListeSSE2_temp)) faire
17:      SSE2 ← obtenirElement(ListeSSE2_temp,1)
18:      supprimer(ListeSSE2_temp,1)
19:      si Schémas_Électroniques_Identiques(SSE1, SSE2) alors
20:        a_trouvé_SE_associé ← vrai
21:        ListeSSE2 ← SupprimerElement(ListeSSE2, SSE2)
22:      fin si
23:    fin tant que
24:    si non a_trouvé_SE_associé alors
25:      sont_identiques ← faux
26:    fin si
27:  fin tant que
28:  retourner (sont_identiques et estVide(ListeSSE2))
29: fin si
```

#### 5.3.1.3 Type de composants

Une autre particularité des circuits électroniques est la différenciation entre les portes logiques. En effet, une porte logique peut représenter une « entrée », une « sortie » ou un autre « composant logique », par exemple : *and*, *or*, *xor*, etc. Nous allons donc introduire la notion de « type » pour les portes logiques, ce qui nous permettra de n'effectuer que les transformations d'un composant par un autre composant du même type :

- le type « *Entrée* » pour les portes logiques de type « entrée » ;
- le type « *Sortie* » pour les portes logiques de type « sortie » ;
- le type « *PorteLogique* » pour les autres portes logiques.

Ceci nous permettra d'avoir un gain de temps relatif pendant l'exécution de l'algorithme de similarité entre schémas électroniques.

### 5.3.2 Proposition d'une mesure de similarité

Nous allons maintenant proposer une mesure de similarité entre schémas électroniques basée sur la distance d'édition de graphes. Comme les schémas électroniques peuvent être représentés par des graphes « étiquetés orientés acycliques », et que ceux que nous manipulant sont de petites tailles[7], nous proposons un algorithme pour mesurer la similarité entre schémas électroniques basé sur la distance d'édition de graphes proposée par Bunke (Bunke et Allerman, 1983; Messmer et Bunke, 1998; Neuhaus *et al.*, 2006; Riesen et Bunke, 2008).

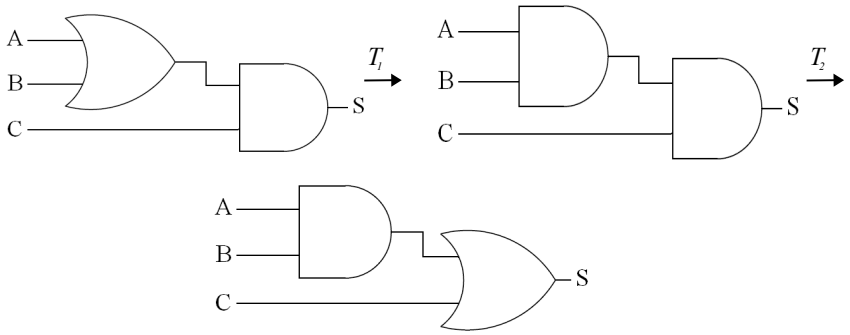



Fig. 5.8 – Exemple d'application de l'algorithme « Graph Edit Distance » entre schémas électroniques

Le principe est d'effectuer une série d'opérations sur le schéma d'origine : ajout, suppression et substitution de composants ou de connexions, afin de le transformer en un autre. Si nous prenons le premier schéma électronique de la figure 5.8, nous pouvons effectuer progressivement les transformations $T_1$ (substitution de *or* par *and*) et $T_2$ (Substitution de *and* par *or*) pour aboutir au troisième schéma électronique de cette figure. Chaque transformation a un coût ($c(T_1)$ est le coût de substitution du composant *or* par le composant *and* et $c(T_2)$ est

[7] En pratique, les exercices de synthèse de circuits proposés aux apprenants sont à vocation pédagogique et donc les schémas obtenus ne dépassent pas généralement une dizaine de composants.

le coût de substitution du composant *and* par le composant *or*), le côut total est la somme des coûts de chaque transformation. Il s'avère dans cet exemple que le coût total est aussi la distance d'édition entre les deux schémas électroniques : $d_{SE}(schema_1, schema_2) = c(T_1) + c(T_2)$.

### 5.3.3 Algorithme de mesure de similarité entre schémas électroniques

Pour l'élaboration de notre algorithme de mesure de similarité entre schémas électroniques, nous nous sommes basés sur les éléments suivants :

- l'algorithme de la distance d'édition entre graphes présenté dans la section 5 page 90 ;
- la notion d'équivalence entre schémas électroniques ;
- et la notion de « type » pour les composants logiques.

**Algorithme 9** Mesure de similarité entre schémas électroniques : $d_{SE}$

**Données:**

$SE_1$ et $SE_2$ deux schémas électroniques non vides.

**Résultats:**

$d_{SE}$, la mesure de similarité entre $SE_1$ et $SE_2$.
$d_{SE}$ est égale à la somme des coûts de toutes les transformations de $p_{min} = \{u_1 \rightarrow v_3, u_2 \rightarrow \varepsilon, ..., \varepsilon \rightarrow v_6\}$.

1: $CL_1 \leftarrow$ obtenirListeCL($SE_1$) // $CL_1 = \{u_1, ..., u_{|CL_1|}\}$, composants logiques de $SE_1$
2: $CL_2 \leftarrow$ obtenirListeCL($SE_2$) // $CL_2 = \{v_1, ..., v_{|CL_2|}\}$, composants logiques de $SE_2$
3:
4: $AR \leftarrow \emptyset$
5: **pour chaque** $w \in CL_2$ **faire**
6: **si** typeCL($w$)=typeCL($u_1$) si les deux composants sont du même type **alors**
7: **insérer** la substitution $\{u_1 \rightarrow w\}$ dans $AR$
8: **fin si**
9: **fin pour**
10: **insérer** la suppression $\{u_1 \rightarrow \varepsilon\}$ dans $AR$
11:
12: **répéter**
13: **enlever** $p_{min} = argmin_{p \in AR}\{g(p)\}$ de $AR$
14: $SE_{1bis} \leftarrow$ **appliquer** les transformations de $p_{min}$ à $SE_1$
15: **si** (**non** Schémas_Électroniques_Équivalents($SE_{1bis}$,$SE_2$)) **alors**
16: soit $p_{min} = \{u_1 \rightarrow v_{i_1}, ..., u_k \rightarrow v_{i_k}\}$
17: **si** $k < |CL_1|$ **alors**
18: **pour chaque** $w \in CL_2 \backslash \{v_{i_1}, ..., v_{i_k}\}$ **faire**
19: **si** typeCL($w$)=typeCL($u_{k+1}$) si les deux composants sont du même type **alors**
20: **insérer** $p_{min} \cup \{u_{k+1} \rightarrow w\}$ dans $AR$
21: **fin si**
22: **fin pour**
23: **insérer** $p_{min} \cup \{u_{k+1} \rightarrow \varepsilon\}$ dans $AR$
24: **sinon**
25: **insérer** $p_{min} \cup \bigcup_{w \in V_2 \backslash \{v_{i_1}, ..., v_{i_k}\}} \{\varepsilon \rightarrow w\}$ dans $AR$
26: **fin si**
27: **fin si**
28: **jusqu'à ce que** (Schémas_Électroniques_Équivalents($SE_{1bis}$,$SE_2$))

29: $d_{SE} \leftarrow \sum_{e_i \in p_{min}} c(e_i)$

30: **retourner** $d_{SE}$

Dans notre algorithme de similarité entre schémas électroniques, nous avons utilisé une valeur de h(p) égale à zéro, ce qui revient à ne pas utiliser d'heuristiques dans l'estimation du coût pour les chemins d'édition.

### 5.3.4 Choix des coûts

Le choix des valeurs pour les côuts à utiliser dans l'algorithme de la distance d'édition de graphes est une étape importante. Il dépend du domaine d'application. Nous avons donc choisi les coûts suivants pour chaque type d'opération :

- les coûts de suppression et d'insertion d'un composant $n$ doivent être $> 0$, nous avons choisi les valeurs : $c_{nd}(n) = c_{ni}(n) = 1$ ;
- Le coût de substitution d'un composant est : $c_{ns}(n, v) = 2$ si les composants $n$ et $v$ ont des étiquettes différentes, $c_{ns}(n, v) = 0$ si les étiquettes des composants $n$ et $v$ sont identiques. En effet, l'opération de substitution de deux composants ayant des étiquettes différentes correspond à effectuer une opération de suppression (du premier composant), suivie d'une opération d'insertion (du deuxième composant), d'où la valeur choisie de 2 ;
- les coûts de suppression et d'insertion de l'arc $i$ sont : $c_{ed}(i) = c_{ei}(i) = 1$ ;
- les connexions d'un schéma électronique n'ont pas d'étiquettes. Pour le coût de substitution, nous allons utiliser le sens de la connexion de telle façon que : $c_{es}(i, j) = 2$ si les directions des connexions $i$ et $j$ sont opposées, $c_{es}(i, j) = 0$ si les directions des connexions $i$ et $j$ sont identiques.

## 5.4 Validation à l'aide de données réelles

### 5.4.1 Description des données

Afin de valider cette mesure de similarité nous l'avons confrontée à des données réelles (Tanana *et al.*, 2008b). Nous avons récupéré le résultat de travaux dirigés et examens corrigés de plusieurs promotions d'élèves ingénieurs. Il s'agit d'exercices de synthèse d'un circuit logique, proposés à des élèves de 3$^{e}$ année d'école d'ingénieurs ou équivalent.

Les exercices proposés sont :

- Additionneur binaire avec retenue (64 productions d'apprenants avec schéma),
- Soustracteur binaire avec retenue (62 productions d'apprenants avec schéma),
- Complément à 2 d'un nombre de 4 bits (16 productions d'apprenants avec schéma).

Nous avons ensuite synthétisé les « corrections manuelles » de l'enseignant, dont un extrait est donné par le tableau 5.1 contenant le détail de chaque étape de la synthèse d'un circuit logique.

| **N° du circuit** | **E/S** | **Table de vérité** | **Simplifications Karnaugh** | **Causes des erreurs** | **Schéma** |
|---|---|---|---|---|---|
| 001 | Ok | Ok | Ok | | Ok |
| 002 | Ok | Ok | Ok | | Ok |
| ... | | ... | ... | ... | |
| 007 | Ok | Ok | Ok | | Ok |
| 019 | Ok | Ok | Ok | Erreur de construction. | $S_i$ : manque un lien entre deux composants,<br>$R_i$ : Ok. |
| ... | | ... | ... | ... | |
| 111 | Ok | $S_i$ : la combinaison 011 n'est pas correcte,<br>$R_i$ : la combinaison 011 n'est pas correcte. | $S_i$ : Erreur dans la table de vérité,<br>$R_i$ : Erreur dans la table de vérité. | Erreurs dans la table de vérité. | $S_i$ : Erreur dans la table de vérité,<br>$R_i$ : Erreur dans la table de vérité. |
| ... | | ... | ... | ... | |

Tab. 5.1 – Extrait du tableau de synthèse des corrections manuelle pour l'additionneur binaire avec retenue : Annexe A.1

### 5.4.2 Description et interprétation des résultats

Pour cette première expérimentation, nous avons pris un seul exercice : « l'additionneur binaire avec retenue ». Il est constitué de 3 entrées ($A_i, B_i$ et $R_{i-1}$) et de 2 sorties ($S_i$ et $R_i$). Nous avons ensuite procédé à un classement manuel des différents schémas des apprenants pour cet exercice. Les schémas les mieux simplifiés sont classés au premier rang représenté par « 1 », les non simplifiés sont classés les derniers, au rang « 10 » (leur fonction logique n'a pas du tout été simplifiée). Les autres sont classés entre 1 et 10, suivant les simplifications appliquées par l'apprenant et les appréciations de l'enseignant. Les schémas « incorrects » n'ont pas été classifiés manuellement, juste l'attribut « Incorrect » leur a été affecté pour le moment.

Nous avons ensuite comparé l'évaluation de l'enseignant avec la mesure de similarité entre schémas électroniques. Pour cela, nous avons choisi un seul schéma de référence par sortie (celui que l'enseignant estime qu'il doit être classé comme 1$^{er}$ de la liste), et nous avons effectué une « évaluation sommative » des productions des apprenants par rapport à

ce schéma de référence. Tous les circuits des apprenants ont été comparés à ce même circuit de référence, même ceux dont la table de vérité est considérée comme « incorrecte ».

Les résultats obtenus pour la sortie $S_i$ sont affichés dans le tableau 5.2.

| Schéma | Classement manuel | Mesure de Similarité | Schéma | Classement manuel | Mesure de Similarité | Schéma | Classement manuel | Mesure de Similarité |
|---|---|---|---|---|---|---|---|---|
| 001 | 5 | 17 | 025 | 1 | 0 | 117 | 1 | 0 |
| 002 | 3 | 6 | 026 | 3 | 6 | 118 | Incorrect | 29 |
| 003 | 1 | 0 | 027 | 5 | 19 | 119 | 3 | 6 |
| 004 | 1 | 0 | 028 | 1 | 0 | 121 | 3 | 6 |
| 005 | 1 | 0 | 100 | 5 | 19 | 122 | 3 | 6 |
| 006 | 1 | 0 | 101 | 5 | 19 | 123 | 3 | 6 |
| 007 | 5 | 19 | 102 | 1 | 4 | 124 | 1 | 4 |
| 008 | 5 | 17 | 103 | 1 | 0 | 125 | 1 | 4 |
| 009 | 5 | 17 | 104 | 1 | 0 | 128 | 3 | 6 |
| 010 | 1 | 0 | 105 | 3 | 6 | 131 | 1 | 4 |
| 011 | 3 | 6 | 106 | 5 | 19 | 136 | 1 | 0 |
| 012 | 5 | 17 | 107 | 1 | 0 | 137 | 1 | 4 |
| 013 | 1 | 0 | 108 | 4 | 16 | 139 | 1 | 0 |
| 014 | 1 | 0 | 109 | 1 | 4 | 140 | 3 | 6 |
| 015 | 1 | 0 | 110 | 2 | 4 | 141 | Incorrect | 23 |
| 017 | 1 | 4 | 111 | Incorrect | 28 | 142 | 3 | 6 |
| 018 | 1 | 4 | 112 | 1 | 0 | 143 | 1 | 0 |
| 019 | Incorrect | 1 | 113 | 3 | 6 | 144 | 1 | 0 |
| 020 | 1 | 0 | 114 | 1 | 4 | 145 | 3 | 6 |
| 021 | 1 | 0 | 115 | 3 | 6 | 146 | 1 | 0 |
| 023 | 5 | 17 | 116 | 10 | 27 | 147 | 1 | 4 |
| 024 | 1 | 0 | | | | | | |

TAB. 5.2 – Résultats de la comparaison de la sortie $S_i$ avec un seul exemple de référence

Afin d'interpréter les résultats de l'évaluation sommative du tableau 5.2, nous avons calculé, uniquement pour les schémas corrects, la moyenne et l'écart-type entre le classement manuel et la distance d'édition entre schémas électroniques. Les résultats sont présentés dans le tableau 5.3 : la première colonne représente le classement manuel des schémas rencontrés dans notre échantillon des productions des apprenants. La deuxième colonne représente la moyenne des mesures de similarité pour un classement donné de la sortie $S_i$, et la troisième colonne, l'écart-type pour ce même classement. Enfin, les quatrième et cinquième colonnes donnent les mêmes informations pour la sortie $R_i$.

| | $S_i$ | | $R_i$ | |
|---|---|---|---|---|
| Classement Manuel | Moyenne | Écart-type | Moyenne | Écart-type |
| 1 | 1,21 | 1,87 | 1,5 | 2,8 |
| 2 | 4 | 0 | 7,08 | 0,34 |
| 3 | 6 | 0 | - | - |
| 4 | 16 | 0 | - | - |
| 5 | 18 | 1,05 | - | - |
| 10 | 27 | 0 | - | - |

TAB. 5.3 – Moyenne et écart-type entre le classement manuel et $d_{SE}$

Ces premiers résultats nous ont permis d'effectuer une évaluation sommative des schémas des apprenants. Les résultats des schémas « corrects » ont été utilisés pour comparer le classement manuel et la distance d'édition entre schémas électroniques. Le cas des schémas « incorrects » ne peut pas être interprété pour le moment, sauf pour les schémas « incorrects » avec une distance courte. Dans tous les cas, nous pouvons faire les constatations suivantes :

1. Les valeurs des écart-types du tableau 5.3 permettent de confirmer que notre mesure de similarité respecte bien le classement manuel de l'enseignant. Par rapport au seul schéma de référence, la mesure de similarité est plus courte pour les schémas qui ont été « bien » classés manuellement.
2. Certains circuits ont été classés de façon identique par l'enseignant mais leurs distances de similarité avec le schéma de référence sont différentes (voir dans le tableau 5.2 les mesures de similarité différentes pour les circuits classés premiers). Ceci est dû au fait que l'enseignant peut très bien considérer que deux schémas différents peuvent être classés de manière identique, du fait qu'ils ont le même « degré » de simplification (une fonction logique pouvant être simplifiée de plusieurs façons différentes).
3. Pour les circuits « incorrects », nous n'avons pas pris en considération, dans la mesure de similarité, le fait que leurs tables de vérité soient incorrectes. Nous pouvons toutefois remarquer, sans faire de conclusions hâtives, qu'ils sont assez éloignés du circuit de référence. Par ailleurs, un cas particulier se distingue, le circuit n°019, qui a une table de vérité « incorrecte », a été classé à une distance de « 1 » du circuit de référence. L'analyse de ce schéma nous a permis de constater qu'il manquait juste une « connexion entre deux composants » par rapport au schéma de référence de l'enseignant. Dès lors, un schéma d'apprenant dont la table de vérité serait fausse, mais avec une mesure de similarité qui serait faible par rapport à un schéma de référence correct, pourrait être considéré comme « faux avec erreur(s) survenue(s) lors de la dernière phase de la synthèse ».

## 5.5 Conclusion

Dans l'expérimentation précédente, nous avons testé l'algorithme de mesure de similarité entre schémas électroniques. Cet algorithme nous a permis d'effectuer une évaluation sommative et normative des productions des apprenants. Il nous faut maintenant réaliser une évaluation formative à partir de la seule réalisation de l'apprenant : son schéma électronique, le seul élément fourni par le logiciel de simulation dont nous disposons. Or, ce résultat découle de toute une démarche intellectuelle qui a été décrite dans la section 4.4 page 70. Il peut donc y avoir des erreurs à n'importe quelle étape de la synthèse et notre objectif est donc de parvenir à repérer l'étape où l'erreur a été commise par l'apprenant. Pour cela, nous proposons d'utiliser un algorithme de classification et une base d'apprentissage.

# Chapitre 6

# Évaluation formative à l'aide d'un algorithme de classification

**Sommaire**

## Résumé

Dans le chapitre 5, nous avons adapté la mesure de similarité entre graphes, proposée par Bunke (Bunke et Allerman, 1983; Messmer et Bunke, 1998; Neuhaus *et al.*, 2006; Riesen et Bunke, 2008), à notre propre contexte pédagogique. Ensuite, nous l'avons validée sur un échantillon de productions des apprenants avec schéma électronique correspondant à l'exercice « additionneur binaire avec retenue ». Cette expérimentation s'est déroulée en plusieurs étapes :

- Classement manuel des productions des apprenants : du plus simplifié (classé 1er), au moins simplifié (classé 10e) ;
- Choix de l'enseignant d'un schéma de référence : celui qu'il estimait le meilleur[1] pour cet exercice ;
- Calcul de la mesure de similarité entre les productions des apprenants et le schéma de référence : même les schémas incorrects ont été comparés au schéma de référence ;
- Validation de la mesure de similarité entre schémas électroniques : par rapport au seul schéma de référence, la mesure de similarité est plus courte pour les schémas qui ont été « bien » classés manuellement.

Ceci nous a permis d'effectuer une évaluation sommative et normative des productions des apprenants. Il nous faut maintenant réaliser une évaluation formative à partir de la seule réalisation de l'apprenant : son schéma électronique, le seul élément fournis par le logiciel de simulation dont nous disposons, et ce à l'aide d'un algorithme de classification supervisée. Pour cela, nous avons besoin de définir une base d'apprentissage et de choisir un algorithme de classification.

Dans ce chapitre, avant de construire une base d'apprentissage contenant des schémas électroniques étiquetés, nous allons d'abord définir le concept d'« étiquette » dans notre propre contexte pédagogique. Nous allons ensuite choisir un algorithme de classification que nous expérimenterons, dans un premier temps, avec une base d'apprentissage construite manuellement. Nous reprendrons le même exercice que dans l'expérimentation précédente. Ensuite, nous recommencerons la même expérience avec une base d'apprentissage générée automatiquement. Finalement, nous étendrons les expérimentations à d'autres exercices et nous terminerons par une interprétation des résultats obtenus.

## 6.1 Identification des étiquettes de données pour une évaluation formative

Notre objectif est d'effectuer l'évaluation formative des schémas électroniques produits par les apprenants en utilisant des algorithmes de classification supervisée. Pour cela, nous

[1] Le plus simplifié ou le plus optimisé.

avons besoin de construire une base d'apprentissage contenant des schémas électroniques « étiquetés ». En effet, dans le chapitre 2, nous avons expliqué que les éléments ou objets qui font partie d'une base d'apprentissage, dans le cas d'une classification supervisée, sont identifiés par une « étiquette » ou « caractéristique ». C'est cette même étiquette qui définit la classe d'appartenance d'un objet dans cette base.

Avant de penser à la création d'une telle base, nous allons d'abord définir comment identifier un schéma électronique « correct » et un schéma électronique « incorrect » avant qu'il ne soit intégré dans cette base.

La première caractéristique qui se dégage pour identifier les schémas électroniques est la « table de vérité ». En effet, les schémas électroniques sont considérés comme corrects si leur table de vérité est correcte, et sont considérés comme incorrects dans le cas contraire. Ensuite, une fois la table de vérité vérifiée, nous nous intéressons au « degré de simplification » de la fonction logique qui est déduite à partir de cette table. En fonction de cette simplification, le schéma électronique obtenu à la fin de la synthèse (cf. section 4.4 page 70) sera plus ou moins « optimisé ».

Dans les sections suivantes, nous allons expliquer comment sont représentés les schémas électroniques et comment nous leur avons attribués des caractéristiques dans le cas des schémas corrects et incorrects.

### 6.1.1 Représentation d'un schéma électronique

Nous avons vu dans la section 4.7.1 page 77 que les schémas électroniques produits par les apprenants ont une représentation « graphique » propre au simulateur que nous avons adopté. Ensuite, pour les besoins de notre plate-forme d'évaluation, nous avons converti ce format graphique en un format logique plus facile à manipuler. Le schéma électronique « logique » obtenu est représenté par un graphe-circuit (cf. section 5.3.1 page 92). Il est par la suite transformé en plusieurs schémas électroniques « logiques », autant que le nombre de sorties du circuit d'origine.

À la représentation « logique » d'un schéma électronique, déjà introduite dans la section 4.7.1 page 77, nous allons ajouter deux nouveaux concepts qui vont nous permettre ensuite de construire la base d'apprentissage des schémas électroniques de référence :

- Le concept « Statut » : qui définit si le schéma électronique est correct ou incorrect. Ceci revient à vérifier la validité de la table de vérité du schéma électronique ;
- Le concept « Infos. Comp. » (Informations complémentaires) : qui représente des informations supplémentaires concernant le degré de simplification du schéma électronique ou le type d'erreurs dans le cas des schémas incorrects.

Ces concepts auront des valeurs particulières suivant si le schéma électronique a une table de vérité correcte ou incorrecte. Les deux concepts réunis représentent « l'étiquette » qui

va identifier le schéma électronique dans la base d'apprentissage. Nous allons donc détailler ces concepts dans les deux sections suivantes, en prenant des exemples concrets.

### 6.1.2 Cas des schémas électroniques corrects

Prenons un schéma électronique « correct », par exemple, le schéma électronique de la sortie $S_i$ du circuit logique d'un additionneur binaire avec retenue :

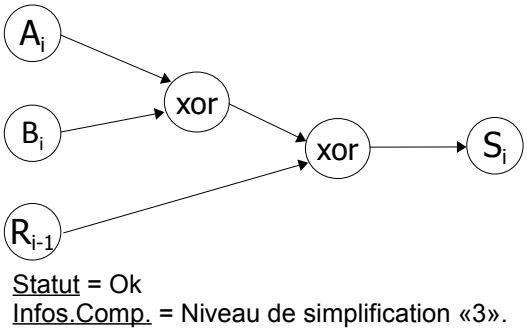



FIG. 6.1 – Exemple d'un schéma électronique correct avec la valeur de son étiquette

Nous distinguons pour ce schéma électronique les caractéristiques suivantes :

- « Statut »=Ok : représente l'état « correct » de la table de vérité du schéma électronique pour la sortie $S_i$ ;
- « Infos. Comp. » : représente le « niveau » ou « degré » de simplification de la fonction logique pour cette sortie. Nous avons choisi le niveau 0, pour une expression logique non simplifiée du tout (contenant tous les termes de la première forme canonique), et les niveaux suivants 1, 2, $\cdots$ , $n$, pour les schémas électroniques ayant leur fonction logique de plus en plus simplifiée jusqu'au niveau $n$. Ce dernier niveau représente les schémas électroniques dont la fonction logique a été simplifiée de façon optimale. La valeur de $n$ dépend de la fonction logique et du nombre d'opérations nécessaires pour la simplifier de façon optimale. Dans le cas de notre exemple $n = 3$.

### 6.1.3 Cas des schémas électroniques incorrects

À la différence des schémas corrects qui ont tous la même table de vérité, les schémas incorrects ont chacun une table de vérité différente qui dépend des erreurs qui se sont produites. Ces erreurs sont occasionnées par l'apprenant à un moment donné durant la synthèse du circuit d'origine. Dans la section 4.8 page 80 nous avons détaillés les différents types d'erreurs qui peuvent être rencontrées. Ces erreurs peuvent être :

- multiples : plusieurs erreurs apparaissent dans la table de vérité ;
- isolées : une seule erreur s'est produite dans la table de vérité ;

– « erreurs-type » : une ou plusieurs « erreurs-type » se sont produites dans la table de vérité, ce qui correspond à des erreurs généralement rencontrées dans la synthèse des circuits logiques. Il va de soi que ces « erreurs-type » dépendent des exercices proposés aux apprenants.

Si nous voulons répertorier toutes les erreurs possibles d'une table de vérité, nous devons effectuer le calcul suivant (Serrell, 1953) : Pour $n$ variables d'entrée différentes, nous pouvons définir $2^{2^n}$ tables de vérité différentes. Une seule correspond à notre fonction logique et les $2^{2^n} - 1$ restantes correspondent aux tables de vérité incorrectes, c'est-à-dire, aux nombres d'erreurs pouvant être commises. Nous voyons donc que ce nombre peut augmenter très rapidement en fonction de la valeur de $n$. Par Exemple, si $n = 1$, le nombre d'erreurs possibles est $2^{2^1} - 1 = 3$, si $n = 4$, le nombre d'erreurs possibles est $2^{2^4} - 1 = 65535$.

Nous voyons bien que le nombre d'erreurs augmente très rapidement en fonction du nombre d'entrées. Il est donc impossible de représenter toutes les erreurs imaginables pour un exemple donné. Par conséquence, nous allons nous limiter à représenter uniquement quelques erreurs-type.

Prenons l'exemple du schéma électronique « incorrect » représenté par le graphe de la figure 6.2.

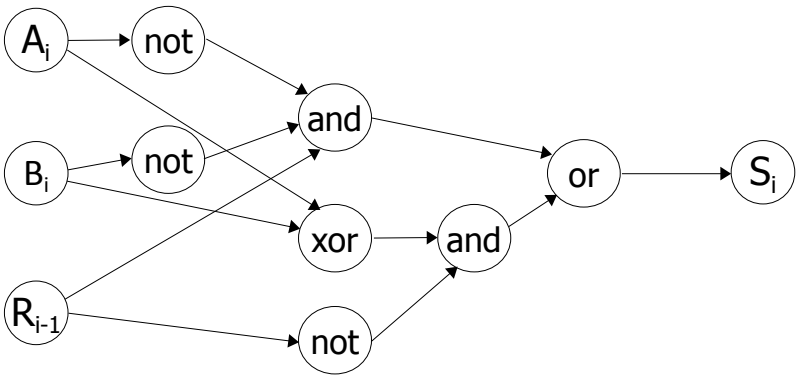



FIG. 6.2 – Exemple d'un schéma électronique incorrect avec la valeur de son étiquette

Ce graphe représente la sortie $S_i$ du circuit logique d'un additionneur binaire avec retenue. Cette sortie du circuit a les caractéristiques suivantes :

– « Statut »=Ko : représente l'état « incorrect » de la table de vérité du schéma électronique pour la sortie $S_i$ ;

– « Infos. Comp. » représente plusieurs informations possibles dans le cas des schémas incorrects :

- indique si c'est une « erreur-type ». Dans notre exemple, c'est l'erreur-type numéro « 8 », correspondant à la combinaison binaire 111 des entrées. Cette erreur indique que l'apprenant n'a pas su ajouter la retenue au résultat de l'addition binaire. Le résultat devant être $1+1+1=1$ retenue 1 alors que l'apprenant a répondu $1+1+1=0$ retenue 1 ou 0,
- indique les combinaisons des entrées pour les différentes erreurs rencontrées. Dans le cas de notre exemple, c'est la combinaison logique $111_2 = 7_{10}$[2]. Dans d'autres exercices, s'il y a plusieurs erreurs, toutes les combinaisons logiques où les erreurs se sont produites sont affichées,
- indique le « niveau » ou « degré » de simplification de la fonction logique représentant la table de vérité pour cette sortie dans le cas d'une « erreur-type »[3]. En effet, même si l'apprenant a commis une erreur dans la table de vérité (pour les erreurs-type), sa démarche de résolution est peut-être correcte. Cette information peut nous renseigner sur le degré de connaissance de l'apprenant vis-à-vis des méthodes de simplifications des fonctions logiques,
- indique si le schéma électronique de l'apprenant est « graphiquement proche » d'un schéma électronique de la base d'apprentissage au sens de la mesure de similarité entre schémas électroniques, même si leurs tables de vérité ne sont pas identiques. Ce genre d'information pourrait indiquer une erreur de construction du schéma électronique.

Finalement, dans le cas des schémas incorrects, l'information contenue dans « Infos. Comp. » peut représenter :

- soit l'appartenance d'un schéma électronique à une catégorie donnée, si cette catégorie est présente dans la base d'apprentissage (par exemple, si le schéma concerné correspond à une erreur-type) ;
- soit le schéma électronique est proche graphiquement d'un autre schéma de la base d'apprentissage ; ce qui pourrait correspondre éventuellement à une erreur de construction ;
- soit une erreur quelconque qui n'est pas identifiée dans la base d'apprentissage.

Dans tous les cas, l'information qui est contenue dans « Infos. Comp. », aussi bien pour les schémas corrects et incorrects, fournit une aide à l'évaluation formative des productions des apprenants. Dans le cas où il subsiste un doute sur le résultat de l'évaluateur automatique, l'enseignant reprend le relais pour prendre une décision finale.

[2] $111_2$ représente, en base binaire, la valeur décimale 7.

[3] Le principe du niveau de simplification est le même que dans le cas d'un schéma correct (cf. section 6.1.2 page 110).

## 6.2 Utilisation d'un algorithme de classification supervisée pour une évaluation formative (semi-)automatique

### 6.2.1 Comment est construite la base d'apprentissage dans notre cas ?

Dans le cas des schémas électroniques, nous avons construit notre base d'apprentissage en se basant sur les critères suivants :

- la table de vérité : un schéma électronique pourra appartenir à une classe ou catégorie de schémas électroniques de référence de la base d'apprentissage à condition qu'il ait la même table de vérité que ces schémas de référence ;
- le degré de simplification de la fonction logique : un schéma électronique pourra appartenir à une classe de schémas électroniques de référence de la base d'apprentissage à condition qu'il soit proche (au sens de la mesure de similarité) d'au moins un schéma électronique de cette classe.

D'une façon générale, nous avons divisé la base d'apprentissage en deux grandes catégories : Les schémas électroniques qui sont corrects et les schémas électroniques qui sont incorrects. Dans le premier cas, tous les schémas ont la même table de vérité et sont donc distingués par le degré de simplification de leur fonction logique. Dans le deuxième cas, les schémas électroniques incorrects sont ensuite regroupés par une autre sous-catégorie, qui est la table de vérité « incorrecte », avant d'être ensuite regroupés par le degré de simplification de leur fonction logique. En somme, un schéma quelconque, produit par un apprenant, ne sera comparé à un autre schéma de la base d'apprentissage que s'ils ont la même table de vérité (correcte ou incorrecte). À l'exception, des schémas incorrects qui ne correspondent pas à des erreurs-type et qui seront donc comparés à tous les schémas de la base d'apprentissage, à la recherche d'éventuelles « erreurs de construction ».

### 6.2.2 Quel algorithme choisir ?

Nous avons vu dans le chapitre 2 que les algorithmes de classification supervisée disponibles sont nombreux. Nous pouvons citer entre autres le k-ppv, les réseaux de neurones, les SVM et les réseaux bayésiens naïfs. Mais, excepté le k-ppv, les autres méthodes ont besoin de beaucoup de données d'apprentissage préalable pour bien fonctionner, or, nous n'en avons que très peu. De plus, nous avons besoin d'interpréter le résultat de l'algorithme de classification en calculant la proximité d'un schéma électronique par rapports aux schémas de référence, tout en rejetant certaines données si la distance obtenue par rapport aux données d'apprentissage est trop grande, chose que les autres algorithmes ne permettent pas de faire.

Nous avons donc choisi d'implémenter l'algorithme du k-ppv en choisissant pour l'instant le cas le plus simple (k=1), c'est-à-dire, le cas où l'étiquette d'une nouvelle donnée sera identique à l'étiquette de la donnée la plus proche dans la base d'apprentissage.

Un algorithme formel du 1-ppv adapté aux schémas électroniques est présenté ci-après :

**Algorithme 10** Algorithme des 1 plus proche schéma électronique : (1-ppvSE)

**Données:**

$SE$ un schéma électronique de l'apprenant ;

$\mathcal{D}_{SE}$ la base d'apprentissage des schémas électroniques de référence.

**Résultats:**

La classe d'appartenance de $SE$ et la mesure de similarité de $SE$ par rapport à cette classe.

1: **pour chaque** $SE_i \in \mathcal{D}_{SE}$ **faire**
2: calculer la mesure de similarité entre $SE_i$ et $SE$ : $d_{SE}(SE_i, SE)$
3: **fin pour**

4: $x_{1-ppvSE} \leftarrow argmin_{SE_i \in \mathcal{D}_{SE}}\ d_{SE}(SE_i, SE)$
5: **retourner** (obtenirÉtiquetteSE($x_{1-ppvSE}$), $d_{SE}(x_{1-ppvSE}, SE)$)

## 6.3 Validation à l'aide d'une base d'apprentissage construite manuellement

Pour cette expérimentation, nous avons repris les mêmes données que la première expérimentation (cf. 5.4.2 page 102), en utilisant les productions des apprenants pour l'additionneur binaire avec retenue (Tanana *et al.*, 2008b).

### 6.3.1 Création des données d'apprentissage

Dans un premier temps, nous avons constitué notre base de « données d'apprentissage » de façon manuelle. Elle est constituée de deux catégories de données : les schémas « corrects » et les schémas « incorrects ». Chaque schéma, de chaque catégorie, est étiqueté suivant son statut et son degré de simplification. C'est cette même étiquette qui nous permettra d'effectuer une évaluation formative des productions des apprenants. En effet, un schéma de l'apprenant se trouvant à une mesure de similarité minimale d'un schéma de référence étiqueté, appartiendra le plus probablement à cette classe de schémas.

Le contenu de cette base pour les exemples « corrects » est le suivant :

- 3 exemples étiquetés « simplification optimale » (Schéma : 01, 02, 03).
- 3 exemples étiquetés « simplification semi-optimale » (Schéma : 04, 05, 06).
- 1 exemple étiqueté « pas de simplification » (Schéma 07).

Et des exemples « incorrects », qui correspondent à une « erreur-type »généralement rencontrée pour l'exercice de l'additionneur :

– 3 exemples étiquetés « simplification optimale - erreur : oublie de la retenue dans la sommation » (Schémas : K01, K02 et K03).
– 3 exemples étiquetés « simplification semi-optimale - erreur : oublie de la retenue dans la sommation » (Schéma : K04, K05, K06).
– 1 exemple étiqueté « pas de simplification - erreur : oublie de la retenue dans la sommation » (Schéma K07).

Le choix de « 7 » exemples par catégorie n'est pas arbitraire. Ces exemples sont choisis de telle façon que nous puissions avoir, sur une échelle de 1 à 10 (correspondant au classement manuel de l'enseignant), un recouvrement suffisant des catégories à représenter.

En effet, sur les 7 exemples des schémas corrects, il y a 3 exemples qui représentent la catégorie « 1 » par rapport à l'échelle de classement de l'enseignant (simplification optimale), 3 exemples qui représentent la catégorie « 5 » (simplification semi-optimale) et 1 seul exemple qui représente la catégorie « 10 » (aucune simplification).

Le même raisonnement correspond aux exemples incorrects, sauf que dans ce cas, nous n'avons représenté qu'une seule erreur-type pour cet exercice.

### 6.3.2 Algorithme de l'évaluation formative des schémas électroniques

Le principe de notre algorithme d'évaluation (voir figure 6.3) se base sur l'algorithme de classification supervisée 1-ppv et sur la mesure de similarité entre schémas électroniques que nous avons défini dans le chapitre 5.

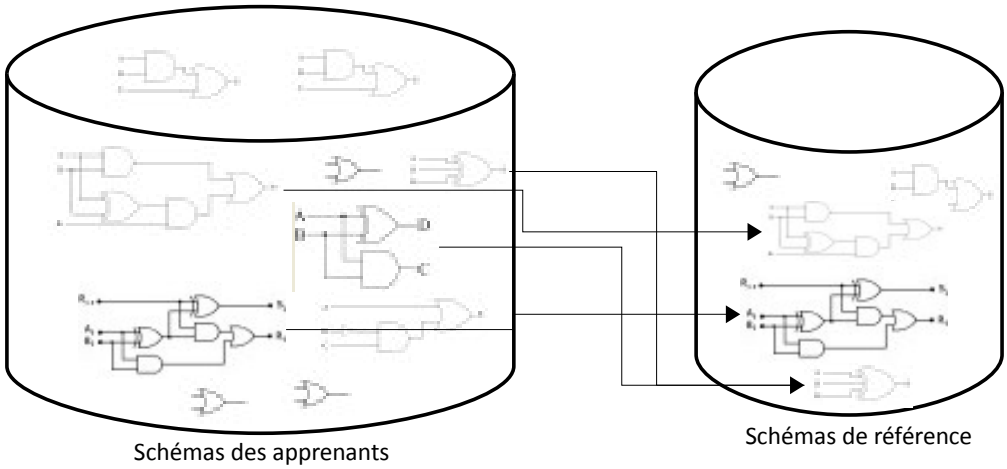



FIG. 6.3 – Principe de fonctionnement de l'algorithme d'évaluation formative

Toutefois, afin de réduire le temps de calcul, nous avons introduit des filtres dans la base d'apprentissage avant d'effectuer le calcul de mesure de similarité. D'un côté, un schéma

de l'apprenant n'est « comparé » à un autre schéma de référence de la base d'apprentissage que s'ils ont la même table de vérité. D'un autre côté, si nous voulons vérifier qu'il y a éventuellement une « erreur de construction » pour un schéma incorrect, nous comparons ce schéma à tous les autres schémas de la base d'apprentissage, mais à condition que la différence des nombres de composants entre les deux schémas est inférieure à un certain nombre de composants [4]. De toute façon, dans ce dernier cas, ce que nous recherchons, ce n'est pas une « équivalence » entre les deux schémas, mais plutôt si l'un des deux schémas est « graphiquement proche » de l'autre (cf. section 6.1.3 page 112).

Finalement, si un schéma incorrect ne correspond à aucun type d'erreurs (erreur identifiée comme « erreur-type » ou de construction), le schéma sera traité comme étant « non identifié », et les erreurs de sa table de vérité seront retournées pour information.

Nous présentons ci-après l'algorithme complet de l'évaluation formative en tenant compte des différents filtres :

**Algorithme 11** Évaluation Formative des Schémas Électroniques

**Données:**

$SE$, représente le schéma électronique de l'apprenant,

$\mathcal{D}_{SE} = \mathcal{D}_{SE_{Ok}} \cup \mathcal{D}_{SE_{Ko-Type1}} ... \cup \mathcal{D}_{SE_{Ko-Typen}}$, la base d'apprentissage des schémas électroniques de référence, $\forall i, j \; \mathcal{D}_{SE_i} \cap \mathcal{D}_{SE_j} = \emptyset$

**Résultats:**

Évaluation formative du schéma électronique.

1: $\mathcal{D}_{SE_{choisi}} \leftarrow \mathcal{D}_{SE_x}$, obtenirTdVSE($\mathcal{D}_{SE_x}$)=obtenirTdVSE(SE) // Ensemble des schémas électroniques étiquetés ayant la même table de vérité que $SE$.
2: **si** $\mathcal{D}_{SE_{choisi}} \neq \emptyset$ **alors**
3: **retourner** 1-ppvSE($SE, \mathcal{D}_{SE_{choisi}}$)
4: **sinon**
5: (Étiquette, Distance) $\leftarrow$ 1-ppvSE($SE, \mathcal{D}_{SE}$)
6: **si** Distance $\leq$ *seuil* **alors**
7: **retourner** (« Erreur de construction : », Étiquette, Distance)
8: **sinon**
9: **retourner** (« Erreurs non identifiées : », obtenirTdVSE($SE$))
10: **fin si**
11: **fin si**

Pour notre expérimentation, nous avons considéré que deux schémas sont « graphiquement proches », s'ils ont au maximum une seule porte logique à deux entrées qui les différencie. Ceci revient à ajouter, à l'un des deux schémas, un composant, plus trois connexions pour

[4] Pour notre expérimentation, nous avons choisi ce nombre égale à 3, car au delà de cette valeur, la mesure de similarité entre les deux schémas augmente rapidement.

les avoir identiques. Ce qui correspond à une mesure de similarité entre les deux schémas, inférieure ou égale à $seuil = 5$ (2 pour la substitution ou insertion d'un composant et 3 pour la substitution ou l'insertion de 3 connexions).

### 6.3.3 Description de l'expérimentation

Dans cette deuxième expérimentation, les résultats que nous obtenons ne correspondent pas à un classement, mais à la mesure de similarité minimale par rapport à un schéma de référence de la base d'apprentissage. Un extrait des résultats obtenus est affiché dans le tableau 6.1. La première colonne représente le nom du circuit, la deuxième colonne le classement manuel de l'enseignant, le même que celui de la première expérimentation, et la troisième colonne représente la mesure de similarité minimale (le 1-ppv) par rapport à un schéma de référence qui est affiché dans la quatrième colonne. Les étiquettes de référence sont référencées par les schémas eux-même (cf. section 6.3.1 page 114).

| Schéma | Classement manuel | Mesure de Similarité | Schéma similaire | Schéma | Classement manuel | Mesure de Similarité | Schéma similaire |
|---|---|---|---|---|---|---|---|
| 001 | 5 | 2 | 04 | 025 | 1 | 0 | 01 |
| 002 | 3 | 6 | 01 | 026 | 3 | 6 | 01 |
| 003 | 1 | 0 | 01 | 027 | 5 | 2 | 06 |
| 004 | 1 | 0 | 01 | 028 | 1 | 0 | 01 |
| 005 | 1 | 0 | 01 | 100 | 5 | 2 | 06 |
| 006 | 1 | 0 | 01 | 101 | 5 | 2 | 05 |
| 007 | 5 | 2 | 05 | 102 | 1 | 0 | 02 |
| 008 | 5 | 2 | 04 | 103 | 1 | 0 | 01 |
| 009 | 5 | 2 | 04 | 104 | 1 | 0 | 01 |
| 010 | 1 | 0 | 01 | 105 | 3 | 6 | 01 |
| 011 | 3 | 6 | 01 | 106 | 5 | 2 | 06 |
| 012 | 5 | 2 | 04 | 107 | 1 | 0 | 01 |
| 013 | 1 | 0 | 01 | 108 | 4 | 7 | 04 |
| 014 | 1 | 0 | 01 | 109 | 1 | 0 | 02 |
| 015 | 1 | 0 | 01 | 110 | 2 | 4 | 01 |
| 017 | 1 | 0 | 02 | 111 | Incorrect | 11 | K02 |
| 018 | 1 | 0 | 02 | 112 | 1 | 0 | 01 |
| 019 | Incorrect | 1 | 01 | 113 | 3 | 6 | 01 |
| 020 | 1 | 0 | 01 | 114 | 1 | 0 | 02 |
| 021 | 1 | 0 | 01 | 115 | 3 | 6 | 01 |
| 023 | 5 | 2 | 04 | 116 | 10 | 0 | 07 |
| 024 | 1 | 0 | 01 | .... | | | |

TAB. 6.1 – Extrait des résultats pour la sortie $S_i$ avec 14 exemples de référence (corrects et incorrects) : Annexe A.4

Les constations que nous pouvons faire sont les suivantes :

1. Les mesures de similarité obtenues ont diminué suite à l'augmentation du nombre de schémas de référence dans la base d'apprentissage. Nous avons donc pu identifier avec « exactitude » (mesure de similarité égale à zéro) plus de schémas corrects (près des 2/3 des schémas corrects) ;
2. Dans les autres cas (mesure de similarité différente de zéro), le schéma sélectionné par l'algorithme du 1-ppvSE se trouve à une distance minimale. Toutefois, puisque la mesure n'est pas égale à zéro, nous ne sommes pas sûrs que c'est la bonne évaluation formative que nous obtenons (par exemple l'évaluation formative du circuit 003 est exacte alors que l'évaluation du circuit 002 est certainement plus discutable). Donc, tout ce que nous pouvons faire comme remarques pour le moment, est que les résultats affichés ont permis de classer, à une mesure de similarité de 2, 1/6 des schémas corrects, et pour les schémas corrects restants, ils ont été classés à une mesure de similarité supérieure à 4 ;
3. Finalement, les mesures de similarité des schémas qui ont une table de vérité incorrecte, ont diminué car le nombre de schémas de référence a augmenté.

### 6.3.4 Conclusion de cette expérimentation

Dans cette partie, nous avons proposé une méthode d'évaluation du savoir-faire de l'apprenant en se basant sur l'algorithme de classification supervisée « 1-ppv », utilisant une base d'apprentissage contenant un nombre réduit d'exemples étiquetés. Nous avons pu identifier, avec exactitude ou de façon très proche, plus du 2/3 des productions « correctes » des apprenants. Pour le 1/3 restant ou les productions « incorrectes », nous n'avons pas pu prendre de décision définitive et ce vu le peu d'exemples étiquetés dans notre base d'apprentissage.

## 6.4 Nécessité d'une construction automatique de la base d'apprentissage

Nous confirmons donc la nécessité d'augmenter le nombre d'exemples de la base d'apprentissage. Mais nous ne pouvons pas continuer à les introduire de façon « manuelle ». Nous avons donc décidé de les produire de façon automatique (Tanana *et al.*, 2008b). Pour cela, nous avons travaillé sur un « générateur de schémas électroniques ».

### 6.4.1 Générateur (semi-)automatique de schémas électroniques

Le principe de fonctionnement du générateur est inspiré de la méthode de Quine-McCluskey (1952; 1956). Ce générateur permet de produire automatiquement un ensemble de schémas électroniques issus d'une même table de vérité. Seul le niveau de simplification

de leur fonction logique de base est différent. Ce niveau varie entre le niveau « non simplifié » (niveau 0) et le niveau le « plus simplifié » (niveau n). Les niveaux intermédiaires correspondent à des simplifications intermédiaires de la fonction logique (Tanana *et al.*, 2009).

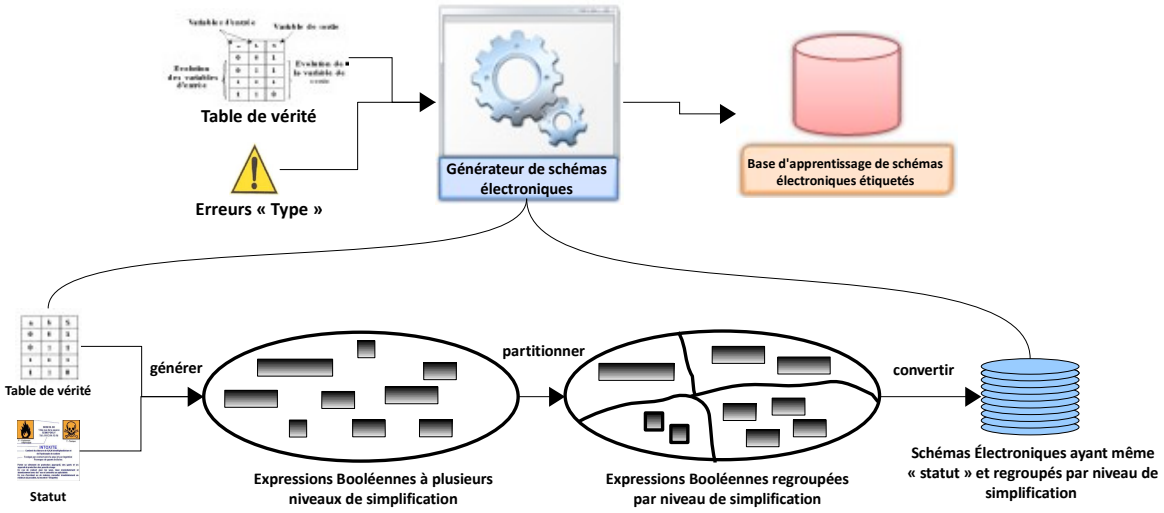



FIG. 6.4 – Principe de fonctionnement du générateur de schémas électroniques

La figure 6.4 explique le fonctionnement du générateur. L'enseignant fournit en entrée une « table de vérité » d'un circuit logique et une liste « d'erreurs-type » correspondant à des erreurs classiques généralement commises pour ce circuit. À partir de la table de vérité, combinée éventuellement à une erreur-type (dans le cas de schémas corrects, aucune erreur n'est transmise au générateur), le générateur génère des « expressions booléennes étiquetées ». Chaque étiquette est composée du « statut » correct ou incorrect de la table de vérité, du type d'erreurs dans le cas des circuits incorrects, et du niveau de simplification de l'expression booléenne. Ensuite, l'ensemble des résultats obtenus est l'union de plusieurs sous-ensembles, regroupant chacun les expressions booléennes ayant les mêmes « étiquettes ». Finalement, les expressions booléennes étiquetées sont converties en schémas électroniques étiquetés. L'ensemble obtenu représente la base d'apprentissage de notre algorithme de classification supervisée.

Le terme (semi-)automatique vient du fait, qu'en plus du générateur automatique décrit ci-dessus, nous rajoutons « manuellement » un autre niveau de simplification (n+1). Ce niveau correspond à l'utilisation d'opérateurs logiques composés, tels que le « *xor* » ou le « *nxor* ». Il correspond aux simplifications particulières telles qu'elles apparaissent dans le schéma explicatif de la synthèse d'un circuit logique dans la figure 4.14 page 80.

L'algorithme suivant correspond au générateur automatique de schémas électroniques, sans la partie introduite manuellement. Pour sa création, nous avons eu besoin de formaliser le type abstrait de données « expressions booléennes étiquetées » dont la définition se trouve en annexe D.

**Algorithme 12** Générateur Automatique de Schémas Électroniques

**Données:**

$TdV$ une table de vérité,

$Statut$ = « Ok » dans le cas d'une table de vérité correcte, = « Ko + erreur-type » dans le cas contraire.

**Résultats:**

Ensemble de schémas électroniques regroupés par statut et par niveau de simplification.

// Étape de génération des expressions booléennes :

1: EBD ← créerEBD($TdV$, $Statut$)
2: Ens_EBD← {EBD}
3: Ens_EBD_Traitées ← ∅

4: **pour chaque** (xEBD ∈ Ens_EBD) **et** (xEBD ∉ Ens_EBD_Traitées) **faire**
5: Ens_EBD_Traitées ← Ens_EBD_Traitées ∪ {xEBD}

6: **pour chaque** xTerme **dans** xEBD **faire**
7: Ens_xTermes ← obtenirTermesAssociésEBD(xEBD, xTerme)
8: Ens_xEBD ← associerTermesEBD(xEBD, xTerme, Ens_xTermes)
9: **fin pour**

10: Ens_EBD ← Ens_EBD ∪ Ens_xEBD
11: **fin pour**

// Étape de partitionnement des expressions booléennes :

12: Ens_EBD_Étiquetées ← ∅
13: **pour chaque** xEBD ∈ Ens_EBD **faire**
14: xEBD ← ajouterEtiquette(xEBD, obtenirStatutEBD(xEBD)+ obtenirNiveauSimplificationEBD(xEBD))
15: Ens_EBD_Étiquetées ← Ens_EBD_Etiquetées ∪ {xEBD}
16: **fin pour**

// Étape de conversion en schémas électroniques :

17: Ens_SE_Étiquetés ← ∅
18: **pour chaque** xEBD ∈ Ens_EBD_Étiquetées **faire**
19: Ens_SE_Étiquetés ← Ens_SE_Étiquetés ∪ {obtenirSE_EBD(xEBD)}
20: **fin pour**

21: **retourner** Ens_SE_Étiquetés

### 6.4.2 Description et interprétation de l'expérimentation

Afin de valider cette troisième expérimentation, nous avons repris les productions des apprenants des deux expérimentations précédentes. Nous avons ensuite exécuté l'algorithme d'évaluation en utilisant :

- l'algorithme de classification « 1-ppvSE », comme précédemment ;
- la mesure de similarité entre schémas électroniques $d_{SE}$ ;
- et la nouvelle base d'apprentissage, plus complète, et qui a été générée automatiquement.

Le tableau 6.2 présente une partie des résultats de cette expérimentation. La première colonne représente le nom du circuit, la deuxième son statut (correct ou incorrect), et les autres colonnes représentent les mesures de similarité du schéma de l'apprenant par rapport à ceux de la base d'apprentissage (« Niv 0 », le schéma de référence non optimisé, et « Niv 3 », les circuits de référence les plus optimaux). Un schéma électronique, dont la mesure de similarité est minimale par rapport à un schéma de référence de niveau « n », appartient le plus probablement à cette catégorie de schémas.

| **Circuit** | **Statut** | **Niveau 0** | **Niveau 1** | **Niveau 2** | **Niveau 3** |
|---|---|---|---|---|---|
| 001 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | [0], 6, 8, 14, 8, 14 | 17, 17 |
| 002 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | [0], 2 |
| 003 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | [0], 8 |
| 007 | Ok | 24 | 17, 17, 19, 17, 17, 19, 11, 13, 19 | 8, 14, 8, 14, [0], 6 | 17, 17 |
| 010 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | [0], 8 |
| 011 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | [0], 2 |
| 012 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | [0], 6, 8, 14, 8, 14 | 17, 17 |
| 019 | Ko | Erreur de construction éventuelle par rapport au circuit correct n° 0 du niveau 3 à une distance de 5 | | | |
| 020 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | [0], 8 |
| 021 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | [0], 8 |
| 023 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | [0], 6, 8, 14, 8, 14 | 17, 17 |
| 026 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | [0], 2 |
| 110 | Ok | 33 | 24, 24, 30, 24, 28, 30, 24, 28, 30 | 15, 21, 19, 21, 19, 21 | 8, [0] |
| 111 | Ko, Type 7 | 18 | 12, 12, 14 | [0] | |
| 116 | Ok | [0] | 9, 11, 13, 9, 11, 13, 9, 11, 13 | 18, 16, 18, 16, 18, 16 | 27, 27 |
| 117 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | [0], 8 |
| 118 | Ko, Type 6 | *[5] | 15, 17, 15 | 15 | |
| 140 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | [0], 2 |
| 141 | Ko | Erreur sur les combinaisons [0,3] | | | |

[5] L'astérisque correspond à une mesure qui a été rejetée par l'évaluateur automatique.

| Circuit | Statut | Niveau 0 | Niveau 1 | Niveau 2 | Niveau 3 |
|---|---|---|---|---|---|
| ... | | | | | |

TAB. 6.2 – Extrait des résultats pour l'évaluation de la sortie $S_i$ de « l'additionneur binaire avec retenue », avec une base d'apprentissage et l'algorithme 1-ppv : Annexe A.6

Dans notre calcul de l'évaluation entre schémas électroniques par rapport à la base d'apprentissage, nous avons considéré que deux schémas sont « graphiquement proches » (donc, une possibilité d'erreur de construction), s'ils ont au maximum une seule porte logique à deux entrées qui les différencie. Ceci revient à ajouter, à l'un des deux schémas, un composant, plus trois connexions pour les avoir identiques, c'est-à-dire, une mesure de similarité entre les deux schémas inférieure ou égale à 5 (= 2 pour la substitution ou insertion d'un composant + 3 pour la substitution ou l'insertion de 3 connexions).

Pour la sortie $S_i$, les résultats obtenus par cette expérimentation sont les suivants : parmi les 64 schémas, 59 ont été évalués « corrects » et identifiés de façon « exacte » dans la base d'apprentissage, 1 schéma évalué « correct » et identifié de façon rapprochée (à une distance de 5 d'un circuit de référence de niveau 2). 3 circuits ont été évalués « incorrects », 1 schéma a été identifié de façon exacte, un autre correspondait à une erreur de construction (Circuit 019) et le dernier n'a pas pu être identifié. Ce qui nous donne un taux d'évaluation avec identification « exacte » dans la base d'apprentissage de 94%. Pour la sortie $R_i$, le taux d'identification « exacte » est à 92%.

### 6.4.3 Extension à d'autres exercices

Nous avons ensuite étendu notre expérimentation à deux autres exercices : le soustracteur binaire avec retenue, 62 productions d'apprenants d'un circuit logique à 3 entrées et 2 sorties, et le complément à 2, 16 productions d'apprenants d'un circuit logique à 4 entrées et 4 sorties. Les résultats complets de toutes les sorties pour tous les exercices se trouvent en Annexe B pour le soustracteur binaire et en Annexe C pour le complément à 2. Pour interpréter ces résultats, nous allons les représenter dans un tableau, pour chaque exercices, par statut correct ou incorrect des schémas des apprenants.

#### 6.4.3.1 Soustracteur binaire avec retenue :

– Pour les schémas corrects :

| Sortie | Identification Exacte | Pas d'Identification | Taux global | Taux partiel |
|---|---|---|---|---|
| $S_i$ | 45 | 0 | 73% | 100% |
| $R_i$ | 24 | 10 | 39% | 71% |

TAB. 6.3 – Les schémas électroniques avec Statut = « Ok ».

La première colonne représente le nom de la sortie. La deuxième colonne, le nombre de schémas corrects qui ont été identifiés de « façon exacte »[6] dans la base d'apprentissage. La troisième colonne représente le pourcentage d'identification des schémas corrects par rapport à **toute** la base d'apprentissage, et la dernière, le pourcentage d'identification des schémas corrects par rapport **uniquement** aux schémas corrects de la base d'apprentissage.

– Pour les schémas inccorrects :

| Sortie | Erreur-Type | Erreur de Construction | Erreur non Identifiée | Taux global | Taux partiel |
|---|---|---|---|---|---|
| $S_i$ | 6 | 6 | 5 | 19% | 71% |
| $R_i$ | 7 | 5 | 16 | 19% | 43% |

TAB. 6.4 – Les schémas électroniques avec Statut = « Ko ».

La première colonne représente le nom de la sortie. La deuxième et la troisième colonne correspondent aux nombre de schémas incorrects pour les erreurs-type et les erreurs de construction. La quatrième colonne, le nombre de schémas incorrects non identifiés dans la base d'apprentissage. Finalement, les deux dernières colonnes représentent les pourcentages sur l'hypothèse de l'origine des erreurs, par rapport à toute la base d'apprentissage et par rapport uniquement aux schémas incorrects de la même base, respectivement.

#### 6.4.3.2 Complément à 2

Nous allons détailler de la même façon que précédemment, les résultats des sorties du complément à 2 :

– Pour les schémas corrects :

| Sortie | Identification Exacte | Pas d'Identification | Taux global | Taux partiel |
|---|---|---|---|---|
| $S_1$ | 15 | 0 | 94% | 100% |
| $S_2$ | 15 | 0 | 94% | 100% |
| $S_3$ | 6 | 6 | 38% | 50% |
| $S_4$ | 6 | 3 | 38% | 67% |

TAB. 6.5 – Les schémas électroniques avec Statut = « Ok ».

– Pour les schémas inccorrects :

[6] Dans nos expérimentations, nous considérons qu'un schéma est identifié de « façon exacte », s'il existe un schéma de référence telle que la mesure de similarité des deux schémas est inférieure ou égale à un certain $seuil = 5$. Le même que celui pour les schémas graphiquement proches.

| Sortie | Erreur-Type | Erreur de Construction | Erreur non Identifiée | Taux global | Taux partiel |
|---|---|---|---|---|---|
| $S_1$ | 1 | 0 | 0 | 6% | 100% |
| $S_2$ | 1 | 0 | 0 | 6% | 100% |
| $S_3$ | 0 | 0 | 4 | 0% | 0% |
| $S_4$ | 4 | 0 | 3 | 25% | 57% |

TAB. 6.6 – Les schémas électroniques avec Statut = « Ko ».

#### 6.4.3.3 Conclusion pour ces expérimentations

Finalement, si nous reprenons les résultats pour tous les exercices confondus, nous obtenons les résultats suivants :

- Le taux d'identification exacte sur toute la base d'apprentissage est de l'ordre de 70% ;
- Le taux d'identification des schémas corrects par rapport à tous les autres schémas corrects de la base est de 91% ;
- Pour les schémas incorrects, nous arrivons à poser des hypothèses sur l'origine des erreurs dans à peu près 56% des cas.

## 6.5 Conclusion

Dans ce chapitre nous avons commencé par effectuer une deuxième expérimentation de l'évaluation formative des productions des apprenants en utilisant la mesure de similarité entre schémas électroniques et une base d'apprentissage créée manuellement par l'enseignant. Nous avons pu identifier, avec exactitude ou de façon très proche, plus du 2/3 des productions « correctes » des apprenants. Pour les productions « incorrectes », aucune hypothèse n'a été établie à cette expérimentation. Nous avons donc conclu à la nécessité d'augmenter le nombre d'exemples dans la base d'apprentissage. Or, comme nous ne pouvions pas continuer à introduire « manuellement » des exemples, nous avons proposé un générateur automatique de schémas électroniques.

Ce générateur permet de produire automatiquement des « bons » et aussi des « mauvais » exemples. Les bons exemples ont été générés et étiquetés en fonction du degré de simplification de leur fonction logique, et les mauvais exemples ont été générés et étiquetés en fonction d'erreurs classiques étudiées lors de la synthèse d'un circuit logique et des optimisations ou simplifications réalisées pour leur construction.

Nous avons ensuite repris les productions des apprenants pour l'exercice de l'additionneur binaire pour une nouvelle expérimentation. Les résultats obtenus ont confirmés une nette augmentation dans le taux d'identification (pour les circuits corrects 95% au lieu de 66% précédemment). Ensuite, nous avons étendu notre expérimentation à deux autres exercices.

De façon générale, le taux d'identification des circuits corrects est très satisfaisant, il est au alentour de 91%. Pour les circuits incorrects, nous arrivons à poser des hypothèses sur l'origine des erreurs dans à peu près 56% des cas. Si nous considèrons le fait que cet outil est proposé à l'enseignant comme une « aide à la correction » des productions des apprenants, les résultats sont alors très satisfaisants.

# Conclusion et Perspectives

# Conclusion et Perspectives

Nos travaux de recherche avaient pour objectif de proposer une méthode d'évaluation formative du « savoir-faire » de l'apprenant basée sur l'utilisation des algorithmes de classification supervisée. Pour cela, nous avons choisi comme domaine d'application l'électronique numérique, plus précisément, l'évaluation de schémas électroniques numériques produits par les apprenants pendant des séances de travaux pratiques. Cet outil est destiné à faciliter la tâche d'évaluation des apprenants à l'enseignant. C'est plus une « aide à la correction » qu'une « évaluation automatique ».

Dans ce dernier chapitre, nous allons faire un dernier bilan en effectuant une synthèse des étapes réalisées durant nos travaux. Ensuite, nous dégagerons les perspectives et les évolutions futures concernant nos travaux.

## Bilan et résultats des travaux

Dans cette thèse, nous sommes partis d'un besoin d'étendre nos travaux sur l'évaluation de l'apprenant en utilisant les algorithmes de classification, du niveau « connaissance », selon la taxonomie de Bloom, au niveau « application », toujours selon cette taxonomie. L'évaluation de ce niveau cognitif correspond à l'évaluation du « savoir-faire » de l'apprenant. Pour cela, nous avons adopté une démarche scientifique reposant sur des cycles « hypothèse-expérimentation-validation ». Chaque cycle est constitué des étapes opératoires suivantes :

- Émettre une hypothèse ;
- Développer les outils nécessaires pour sa vérification ;
- Expérimenter avec des données du domaine et récupérer les résultats fournis ;
- Vérifier et interpréter les résultats pour éventuellement valider l'hypothèse de départ.

Nous avons ainsi opéré en trois cycles successifs. La validation de chaque cycle était nécessaire pour pouvoir poser l'hypothèse suivante.

Le premier cycle correspond à la validation d'une mesure de similarité entre schémas électroniques. Cette mesure a été confrontée à des données réelles du domaine pour une première expérimentation. Durant cette phase, nous avons comparé l'évaluation « manuelle »

de l'enseignant avec la mesure de similarité entre schémas électroniques. Les résultats obtenus par la mesure de similarité respectent bien l'évaluation manuelle de l'enseignant.

Pour le deuxième cycle, nous avons choisi l'algorithme de classification des $k$ plus proches voisins (en choisissant le cas le plus simple où $k = 1$) et une base d'apprentissage construite manuellement, composée de schémas électroniques corrects et incorrects correspondant à plusieurs niveaux de simplification. Nous avons ainsi pu identifier de façon très proches plus du 2/3 des productions « correctes » des apprenants.

Par la suite, pour le troisième cycle, nous avons développé un « générateur de schémas électroniques ». Ce générateur permet de produire automatiquement des schémas « corrects » et « incorrects ». Ces schémas incorrects ont été générés et étiquetés en fonction d'erreurs classiques étudiées lors de la synthèse d'un circuit logique. Dans tous les cas, les schémas produits correspondent à plusieurs niveaux de simplification.

Finalement, nous avons repris les productions des apprenants pour une nouvelle expérimentation. Les résultats obtenus ont confirmé une nette augmentation dans le taux d'identification (pour les circuits corrects 95% au lieu de 66% précédemment). Ensuite, nous avons étendu notre expérimentation à deux autres exercices. Le taux d'identification des circuits corrects était au alentour de 91%. Pour les circuits incorrects, nous sommes arrivés à poser des hypothèses sur l'origine des erreurs dans à peu près 56% des cas. Si nous considèrons le fait que cet outil est proposé à l'enseignant comme une « aide à la correction » des productions des apprenants, les résultats obtenus sont satisfaisants.

Nous pouvons aussi remarquer que pour les données qui n'ont pas été identifiées ou ont été rejetées par le moteur d'évaluation, l'enseignant a toujours la possibilité de les étiqueter « manuellement » et de les insérer dans la base d'apprentissage pour des évaluations ultérieures. Une manière d'enrichir progressivement la base d'apprentissage avec des exemples « intéressants » et qui n'ont pas pu être générés automatiquement.

D'un autre côté, bien que nous ayons travaillé dans un domaine d'application bien déterminé, nous avons gardé un certain degré de généricité à l'élaboration de notre outil. En effet, parmi nos objectifs de départ, nous avons établi la condition de fournir peu de connaissances à l'élaboration du système et à celle des exercices. Nous pensons donc qu'il est possible de généraliser notre méthode à d'autres domaines où les connaissances peuvent être représentées par des « graphes » ou par une « mesure de similarité » entre les données du domaine avec possibilité de générer des exemples.

## Perspectives

Aujourd'hui, l'outil que nous avons proposé pour l'évaluation des schémas électroniques existe uniquement en « prototype ». Il reste encore des développements et des améliorations à effectuer pour avoir un produit utilisable et diffusable. Pour les évolutions futures

de nos travaux, nous distinguons deux genres de perspectives : des considérations techniques et des perspectives de recherche.

Pour les considérations techniques, il reste essentiellement la continuation des développements informatiques pour passer du prototype à une application utilisable et qui pourra être intégrée dans la plate-forme TEB. Le re-développement des algorithmes, qui sont écrits pour l'instant en langage Python, nécessitera sans doute des adaptations et réajustements pour leur exportation au langage de programmation de la plate-forme TEB (Technologie JEE). Nous préconisons aussi que les développements du composant d'évaluation respectent le standard SCORM, afin de permettre à l'application d'être lue par un « player » SCORM. Ce que nous retrouvons dans la plus part des plates-formes actuelles (comme par exemple, Moodle, Claroline, WebCT, etc.).

Une autre considération technique est l'amélioration des performances et des fonctionnalités de notre outil. Nous pouvons citer celle du générateur automatique de schémas électroniques afin de prendre en compte le niveau de simplification (n+1). Ce niveau utilise les opérateurs logiques composés, tels que le « xor » ou le « nxor ». Il correspond aux simplifications supplémentaires telles qu'elles apparaissent dans le schéma explicatif de la synthèse d'un circuit logique de la figure 4.14 page 80. Ce niveau de simplification pourra être développé à partir des explications qui sont décrites par Turton dans (Turton, 1996).

Pour les perspectives de recherche, un aspect à considérer est celui de l'algorithme de la distance d'édition entre schémas électroniques. Sa complexité actuelle est NP-Complet et nous pouvons réfléchir à étudier d'autres distances entre graphes moins gourmandes en complexité. Par ailleurs, nous pouvons mener une réflexion sur l'utilisation d'autres algorithmes de classification, plus performants en temps de calcul, mais qui permettent aussi d'effectuer « un rejet par distance » (avec utilisation, par exemple, de classifieur « one-class » pour faire du rejet).

De plus, comme nous venons de le voir, nous pensons que notre méthode pourra être généralisée à d'autres domaines. Cette généralisation pourra se faire moyennant quelques développements supplémentaires. Par exemple, nous travaillons actuellement sur l'évaluation du savoir-faire des apprenants dans le domaine de l'algorithmique. Les algorithmes produits par les apprenants pouvant être représentés entre autres par des graphes.

Un autre exemple, et celui des résultats de travaux de thèse de L. Auxepaules2009 (Auxepaules, 2009). Il s'est intéressé à l'analyse et le diagnostic des réponses de l'apprenant dans des activités de modélisation, plus particulièrement, la construction d'un diagramme UML. L'outil de diagnostic qui est proposé se base sur la comparaison et l'appariement entre le diagramme de l'apprenant et un **seul** diagramme de référence fournit par l'enseignant. Ce dernier point est une des limitations actuelles de son outil. Nous pensons que tout diagramme UML peut être représenté par des graphes ayant 3 types de nœuds : *classe, attribut* et *méthode*, et 3 types de liens : *agrégation, composition* et *héritage*. L'algorithme sur la distance d'édition entre graphes (GED) pourrait ainsi être adapté à ce cas afin de calculer une mesure

de similarité entre diagrammes UML. La difficulté ne résiderait alors plus que dans la génération automatique de diagrammes à partir d'un modèle de référence de l'enseignant. Nous pouvons toutefois émettre l'hypothèse que notre méthode basée sur l'utilisation d'erreurs types, pourrait dans ce cadre générer plusieurs diagrammes UML à partir d'un exemple de l'enseignant.

Finalement, une autre perspective serait d'étendre « l'aide à la correction » proposée par notre méthode à « l'aide à l'auto-correction ». En effet, actuellement l'outil propose une aide à la correction à l'enseignant pour l'évaluation formative des productions des apprenants. Nous pouvons réfléchir sur l'évolution de cet outil pour qu'il puisse proposer cette aide directement à l'apprenant dans un processus d'auto-formation.

# Annexes

# Annexe A

# Résultats expérimentaux de l'additionneur binaire avec retenue

## A.1 Corrections manuelle de l'additionneur

| Circuit | E/S | Table de vérité | Simplifications Karnaugh | Causes des erreurs | Schéma |
|---|---|---|---|---|---|
| 001 | Ok | Ok | Ok | | Ok |
| 002 | Ok | Ok | Ok | | Ok |
| 003 | Ok | Ok | Ok | | Ok |
| 004 | Ok | Ok | Ok | | Ok |
| 005 | Ok | Ok | Ok | | Ok |
| 006 | Ok | Ok | Ok | | Ok |
| 007 | Ok | Ok | Ok | | Ok |
| 008 | Ok | Ok | Ok | | Ok |
| 009 | Ok | Ok | Ok | | Ok |
| 010 | Ok | Ok | Ok | | Ok |
| 011 | Ok | Ok | Ok | | Ok |
| 012 | Ok | Ok | Ok | | Ok |
| 013 | Ok | Ok | Ok | | Ok |
| 014 | Ok | Ok | Ok | | Ok |
| 015 | Ok | Ok | Ok | | Pas de schéma |
| 016 | Ok | Ok | Ok | | Pas de schéma |
| 017 | Ok | Ok | Ok | | Ok |
| 018 | Ok | Ok | Ok | | Ok |

| Circuit | E/S | Table de vérité | Simplifications Karnaugh | Causes des erreurs | Schéma |
|---|---|---|---|---|---|
| 019 | Ok | Ok | Ok | $S_i$ : manque un lien entre deux composants (Erreur de construction). | $S_i$ : manque un lien entre deux composants,<br>$R_i$ : Ok. |
| 020 | Ok | Ok | Ok | | Ok |
| 021 | Ok | Ok | Ok | | Ok |
| 022 | Ok | Ok | Ok | | Pas de schéma |
| 023 | Ok | Ok | Ok | | Ok |
| 025 | Ok | Ok | Ok | | Ok |
| 026 | Ok | Ok | Ok | | Ok |
| 027 | Ok | Ok | Ok | | Ok |
| 028 | Ok | Ok | Ok | | Ok |
| 100 | Ok | Ok | Ok | | Ok |
| 101 | Ok | Ok | Ok | | Ok |
| 102 | Ok | Ok | Ok | | Ok |
| 103 | Ok | Ok | Ok | | Ok |
| 104 | Ok | Ok | Ok | | Ok |
| 105 | Ok | Ok | Ok | | Ok |
| 106 | Ok | Ok | Ok | | Ok |
| 107 | Ok | Ok | Ok | | Ok |
| 108 | Ok | Ok | Ok | | Ok |
| 109 | Ok | Ok | Ok | | Ok |
| 010 | Ok | Ok | Ok | | Ok |
| 111 | Ok | $S_i$ : la combinaison 011 n'est pas correcte,<br>$R_i$ : la combinaison 011 n'est pas correcte. | $S_i$ : Erreur dans la table de vérité,<br>$R_i$ : Erreur dans la table de vérité. | Erreurs dans la table de vérité. | $S_i$ : Erreur dans la table de vérité,<br>$R_i$ : Erreur dans la table de vérité. |
| 112 | Ok | Ok | Ok | | Ok |
| 113 | Ok | Ok | Ok | | Ok |
| 114 | Ok | Ok | Ok | | Ok |
| 115 | Ok | Ok | Ok | | Ok |
| 116 | Ok | Ok | Ok | | Ok |
| 117 | Ok | Ok | Ok | | Ok |
| 118 | Ok | Ok | Ok | | $S_i$ : Erreur de construction,<br>$R_i$ : Ok. |

| Circuit | E/S | Table de vérité | Simplifications Karnaugh | Causes des erreurs | Schéma |
|---|---|---|---|---|---|
| 119 | Ok | Ok | $S_i$ : Ok,<br>$R_i$ : les combinaisons 110 et 111 ne sont pas correcte. | Erreur de simplification | $S_i$ : Ok,<br>$R_i$ : Erreur de simplification. |
| 120 | Ok | Ok | Ok | | Pas de schéma |
| 121 | Ok | Ok | Ok | | Ok |
| 122 | Ok | Ok | Ok | | Ok |
| 123 | Ok | Ok | Ok | | Ok |
| 124 | Ok | Ok | Ok | | Ok |
| 125 | Ok | Ok | Ok | | Ok |
| 126 | Ok | Ok | $S_i$ : Erreur de simplification,<br>$R_i$ : Ok. | Erreur de simplification | Pas de schéma |
| 127 | Ok | Ok | Ok | | Pas de schéma |
| 128 | Ok | Ok | Ok | | Ok |
| 129 | Ok | Ok | Ok | | Pas de schéma |
| 131 | Ok | Ok | Ok | | Ok |
| 132 | Ok | Ok | Pas de simplification | | Pas de schéma |
| 133 | Ok | Ok | Pas de simplification | | Pas de schéma |
| 134 | Ok | Ok | Ok | | Ok |
| 135 | Ok | Ok | Pas de simplification | | Pas de schéma |
| 136 | Ok | Ok | Ok | | Ok |
| 137 | Ok | Ok | Ok | | Ok |
| 138 | Ok | Ok | Ok | | Pas de schéma |
| 139 | Ok | Ok | Ok | | Ok |
| 140 | Ok | Ok | Ok | | Ok |
| 141 | Ok | Ok | $S_i$ : Erreur de simplification,<br>$R_i$ : Ok. | Erreur de simplification qui a engendré une erreur dans les combinaisons : 000 et 110 | $S_i$ : Erreur de simplification,<br>$R_i$ : Ok. |
| 142 | Ok | Ok | Ok | $R_i$ : Erreur de construction qui a engendré une erreur dans les combinaisons : 110 et 111 | $S_i$ : Ok,<br>$R_i$ : Erreur de construction. |
| 143 | Ok | Ok | Ok | | Ok |

| Circuit | E/S | Table de vérité | Simplifications Karnaugh | Causes des erreurs | Schéma |
|---|---|---|---|---|---|
| 144 | Ok | Ok | Ok | | Ok |
| 145 | Ok | Ok | Ok | | Ok |
| 146 | Ok | Ok | Ok | | Ok |
| 147 | Ok | Ok | Ok | | Ok |

TAB. A.1 – Tableau de synthèse des corrections manuelles pour l'additionneur binaire avec retenue

## A.2 Résultats de comparaison de la sortie $S_i$ avec un seul schéma de référence

| Circuit | Classement manuel | Mesure de Similarité | Circuit | Classement manuel | Mesure de Similarité |
|---|---|---|---|---|---|
| 001 | 5 | 17 | 106 | 5 | 19 |
| 002 | 3 | 6 | 107 | 1 | 0 |
| 003 | 1 | 0 | 108 | 4 | 16 |
| 004 | 1 | 0 | 109 | 1 | 4 |
| 005 | 1 | 0 | 110 | 2 | 4 |
| 006 | 1 | 0 | 111 | Incorrect | 28 |
| 007 | 5 | 19 | 112 | 1 | 0 |
| 008 | 5 | 17 | 113 | 3 | 6 |
| 009 | 5 | 17 | 114 | 1 | 4 |
| 010 | 1 | 0 | 115 | 3 | 6 |
| 011 | 3 | 6 | 116 | 10 | 27 |
| 012 | 5 | 17 | 117 | 1 | 0 |
| 013 | 1 | 0 | 118 | Incorrect | 29 |
| 014 | 1 | 0 | 119 | 3 | 6 |
| 015 | 1 | 0 | 121 | 3 | 6 |
| 017 | 1 | 4 | 122 | 3 | 6 |
| 018 | 1 | 4 | 123 | 3 | 6 |
| 019 | Incorrect | 1 | 124 | 1 | 4 |
| 020 | 1 | 0 | 125 | 1 | 4 |
| 021 | 1 | 0 | 128 | 3 | 6 |
| 023 | 5 | 17 | 131 | 1 | 4 |
| 024 | 1 | 0 | 136 | 1 | 0 |
| 025 | 1 | 0 | 137 | 1 | 4 |
| 026 | 3 | 6 | 139 | 1 | 0 |
| 027 | 5 | 19 | 140 | 3 | 6 |
| 028 | 1 | 0 | 141 | Incorrect | 23 |

| Circuit | Classement manuel | Mesure de Similarité | Circuit | Classement manuel | Mesure de Similarité |
|---|---|---|---|---|---|
| 100 | 5 | 19 | 142 | 3 | 6 |
| 101 | 5 | 19 | 143 | 1 | 0 |
| 102 | 1 | 4 | 144 | 1 | 0 |
| 103 | 1 | 0 | 145 | 3 | 6 |
| 104 | 1 | 0 | 146 | 1 | 0 |
| 105 | 3 | 6 | 147 | 1 | 4 |

TAB. A.2 – Résultats de la comparaison de la sortie $S_i$ avec un seul exemple de référence

## A.3 Résultats de comparaison de la sortie $R_i$ avec un seul schéma de référence

| Circuit | Classement manuel | Mesure de Similarité | Circuit | Classement manuel | Mesure de Similarité |
|---|---|---|---|---|---|
| 001 | 2 | 7 | 106 | 2 | 7 |
| 002 | 2 | 7 | 107 | 2 | 7 |
| 003 | 1 | 6 | 108 | 2 | 7 |
| 004 | 1 | 0 | 109 | 2 | 7 |
| 005 | 1 | 0 | 110 | 2 | 9 |
| 006 | 1 | 0 | 111 | Incorrect | 13 |
| 007 | 1 | 0 | 112 | 2 | 7 |
| 008 | 2 | 7 | 113 | 1 | 0 |
| 009 | 2 | 7 | 114 | 2 | 7 |
| 010 | 1 | 0 | 115 | 2 | 7 |
| 011 | 1 | 0 | 116 | 1 | 7 |
| 012 | 2 | 7 | 117 | 2 | 7 |
| 013 | 2 | 7 | 118 | 1 | 0 |
| 014 | 1 | 0 | 119 | Incorrect | 13 |
| 015 | 1 | 0 | 121 | 1 | 7 |
| 017 | 1 | 0 | 122 | 1 | 0 |
| 018 | 1 | 0 | 123 | 2 | 7 |
| 019 | 1 | 0 | 124 | 2 | 7 |
| 020 | 1 | 0 | 125 | 1 | 6 |
| 021 | 1 | 0 | 128 | 2 | 7 |
| 023 | 2 | 7 | 131 | 2 | 7 |
| 024 | 1 | 0 | 136 | 2 | 70 |
| 025 | 2 | 7 | 137 | 2 | 7 |
| 026 | 1 | 0 | 139 | 2 | 7 |
| 027 | 2 | 7 | 140 | 2 | 7 |

| Circuit | Classement manuel | Mesure de Similarité | Circuit | Classement manuel | Mesure de Similarité |
|---|---|---|---|---|---|
| 028 | 2 | 7 | 141 | 2 | 7 |
| 100 | 2 | 7 | 142 | Incorrect | 13 |
| 101 | 1 | 0 | 143 | 2 | 7 |
| 102 | 2 | 7 | 144 | 1 | 6 |
| 103 | 2 | 7 | 145 | 1 | 0 |
| 104 | 1 | 7 | 146 | 2 | 7 |
| 105 | 2 | 7 | 147 | 2 | 7 |

TAB. A.3 – Résultats de la comparaison de la sortie $R_i$ avec un seul exemple de référence

## A.4 Résultats de comparaison de la sortie $S_i$ avec une base d'apprentissage construite manuellement

| Circuit | Classement manuel | Mesure de Similarité | Schémas Similaires | Circuit | Classement manuel | Mesure de Similarité | Schémas Similaires |
|---|---|---|---|---|---|---|---|
| 001 | 5 | 2 | 04 | 106 | 5 | 2 | 06 |
| 002 | 3 | 6 | 01 | 107 | 1 | 0 | 01 |
| 003 | 1 | 0 | 01 | 108 | 4 | 7 | 04 |
| 004 | 1 | 0 | 01 | 109 | 1 | 0 | 02 |
| 005 | 1 | 0 | 01 | 110 | 2 | 4 | 01 |
| 006 | 1 | 0 | 01 | 111 | Incorrect | 11 | K02 |
| 007 | 5 | 2 | 05 | 112 | 1 | 0 | 01 |
| 008 | 5 | 2 | 04 | 113 | 3 | 6 | 01 |
| 009 | 5 | 2 | 04 | 114 | 1 | 0 | 02 |
| 010 | 1 | 0 | 01 | 115 | 3 | 6 | 01 |
| 011 | 3 | 6 | 01 | 116 | 10 | 0 | 07 |
| 012 | 5 | 2 | 04 | 117 | 1 | 0 | 01 |
| 013 | 1 | 0 | 01 | 118 | Incorrect | 12 | K02 |
| 014 | 1 | 0 | 01 | 119 | 3 | 6 | 01 |
| 015 | 1 | 0 | 01 | 121 | 3 | 6 | 01 |
| 017 | 1 | 0 | 02 | 122 | 3 | 6 | 01 |
| 018 | 1 | 0 | 02 | 123 | 3 | 6 | 01 |
| 019 | Incorrect | 1 | 01 | 124 | 1 | 0 | 02 |
| 020 | 1 | 0 | 01 | 125 | 1 | 0 | 02 |
| 021 | 1 | 0 | 01 | 128 | 3 | 6 | 01 |
| 023 | 5 | 2 | 04 | 131 | 1 | 0 | 03 |
| 024 | 1 | 0 | 01 | 136 | 1 | 0 | 01 |
| 025 | 1 | 0 | 01 | 137 | 1 | 0 | 02 |
| 026 | 3 | 6 | 01 | 139 | 1 | 0 | 01 |

| Circuit | Classement manuel | Mesure de Similarité | Schémas Similaires | Circuit | Classement manuel | Mesure de Similarité | Schémas Similaires |
|---|---|---|---|---|---|---|---|
| 027 | 5 | 2 | 06 | 140 | 3 | 6 | 01 |
| 028 | 1 | 0 | 01 | 141 | Incorrect | 12 | K02 |
| 100 | 5 | 2 | 06 | 142 | 3 | 6 | 01 |
| 101 | 5 | 2 | 05 | 143 | 1 | 0 | 01 |
| 102 | 1 | 0 | 02 | 144 | 1 | 0 | 01 |
| 103 | 1 | 0 | 01 | 145 | 3 | 6 | 01 |
| 104 | 1 | 0 | 01 | 146 | 1 | 0 | 01 |
| 105 | 3 | 6 | 01 | 147 | 1 | 0 | 02 |

TAB. A.4 – Résultats de la comparaison de la sortie $S_i$ avec une base d'apprentissage construite manuellement

## A.5 Résultats de comparaison de la sortie $R_i$ avec une base d'apprentissage construite manuellement

| Circuit | Classement manuel | Mesure de Similarité | Schémas Similaires | Circuit | Classement manuel | Mesure de Similarité | Schémas similaires |
|---|---|---|---|---|---|---|---|
| 001 | 2 | 0 | 05 | 106 | 2 | 0 | 05 |
| 002 | 2 | 0 | 05 | 107 | 2 | 0 | 05 |
| 003 | 1 | 0 | 03 | 108 | 2 | 0 | 05 |
| 004 | 1 | 0 | 01 | 109 | 2 | 0 | 06 |
| 005 | 1 | 0 | 01 | 110 | 2 | 6 | 05 |
| 006 | 1 | 0 | 01 | 111 | Incorrect | 13 | K06 |
| 007 | 1 | 0 | 01 | 112 | 2 | 0 | 03 |
| 008 | 2 | 0 | 05 | 113 | 1 | 0 | 01 |
| 009 | 2 | 0 | 05 | 114 | 2 | 0 | 06 |
| 010 | 1 | 0 | 01 | 115 | 2 | 0 | 05 |
| 011 | 1 | 0 | 01 | 116 | 1 | 2 | 05 |
| 012 | 2 | 0 | 05 | 117 | 2 | 0 | 05 |
| 013 | 2 | 0 | 05 | 118 | 1 | 0 | 01 |
| 014 | 1 | 0 | 01 | 119 | Incorrect | 11 | K05 |
| 015 | 1 | 0 | 01 | 121 | 1 | 2 | 05 |
| 017 | 1 | 0 | 01 | 122 | 1 | 0 | 01 |
| 018 | 1 | 0 | 01 | 123 | 2 | 0 | 05 |
| 019 | 1 | 0 | 01 | 124 | 2 | 0 | 05 |
| 020 | 1 | 0 | 01 | 125 | 1 | 0 | 04 |
| 021 | 1 | 0 | 01 | 128 | 3 | 6 | 01 |
| 023 | 2 | 0 | 05 | 131 | 2 | 0 | 07 |
| 024 | 1 | 0 | 01 | 136 | 2 | 0 | 05 |

| Circuit | Classement manuel | Mesure de Similarité | Schémas Similaires | Circuit | Classement manuel | Mesure de Similarité | Schémas similaires |
|---|---|---|---|---|---|---|---|
| 025 | 2 | 0 | 05 | 137 | 2 | 0 | 06 |
| 026 | 1 | 0 | 01 | 139 | 2 | 0 | 05 |
| 027 | 2 | 0 | 05 | 140 | 3 | 6 | 01 |
| 028 | 2 | 0 | 05 | 141 | 2 | 0 | 06 |
| 100 | 2 | 0 | 05 | 142 | Incorrect | 11 | K05 |
| 101 | 1 | 0 | 01 | 143 | 2 | 0 | 05 |
| 102 | 2 | 0 | 06 | 144 | 1 | 0 | 02 |
| 103 | 2 | 0 | 05 | 145 | 1 | 0 | 01 |
| 104 | 1 | 2 | 05 | 146 | 2 | 0 | 05 |
| 105 | 2 | 0 | 05 | 147 | 2 | 0 | 06 |

TAB. A.5 – Résultats de la comparaison de la sortie $R_i$ avec une base d'apprentissage construite manuellement

## A.6 Résultats de comparaison de la sortie $S_i$ avec une base d'apprentissage construite semi-automatiquement

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 |
|---|---|---|---|---|---|
| 001 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 002 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 003 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 004 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 005 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 006 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 007 | Ok | 24 | 17, 17, 19, 17, 17, 19, 11, 13, 19 | 8, 14, 8, 14, 0, 6 | 17, 17 |
| 008 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 009 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 010 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 011 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 012 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 013 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 014 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 015 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 017 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 018 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 019 | Ko | Proche du circuit correct numéro 0, niveau de simplification 3, à une distance de 5 | | | |
| 020 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 021 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 |
|---|---|---|---|---|---|
| 023 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 024 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 025 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 026 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 027 | Ok | 24 | 17, 17, 19, 11, 13, 19, 17, 17, 19 | 8, 14, 0, 6, 8, 14 | 17, 17 |
| 028 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 100 | Ok | 24 | 17, 17, 19, 11, 13, 19, 17, 17, 19 | 8, 14, 0, 6, 8, 14 | 17, 17 |
| 101 | Ok | 24 | 17, 17, 19, 17, 17, 19, 11, 13, 19 | 8, 14, 8, 14, 0, 6 | 17, 17 |
| 102 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 103 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 104 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 105 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 106 | Ok | 24 | 17, 17, 19, 11, 13, 19, 17, 17, 19 | 8, 14, 0, 6, 8, 14 | 17, 17 |
| 107 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 108 | Ok | 23 | 16, 12, 18, 16, 16, 18, 16, 16, 18 | 5, 9, 9, 15, 9, 15 | 18, 16 |
| 109 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 110 | Ok | 33 | 24, 24, 30, 24, 28, 30, 24, 28, 30 | 15, 21, 19, 21, 19, 21 | 8, 0 |
| 111 | Ko, Erreur Type : 7 | 18 | 12, 12, 14 | 0 | |
| 112 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 113 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 114 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 115 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 116 | Ok | 0 | 9, 11, 13, 9, 11, 13, 9, 11, 13 | 18, 16, 18, 16, 18, 16 | 27, 27 |
| 117 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 118 | Ko, Erreur Type : 6 | * | 15, 17, 15 | 15 | |
| 119 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 121 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 122 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 123 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 124 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 125 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 128 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 131 | Ok | 33 | 24, 28, 30, 24, 26, 28, 24, 28, 30 | 19, 19, 17, 19, 19, 19 | 0, 8 |
| 136 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 137 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 139 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 140 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 |
|---|---|---|---|---|---|
| 141 | Ko | Erreurs sur les combinaisons [0,3] | | | |
| 142 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 143 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 144 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 145 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 146 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 147 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |

TAB. A.6 – Résultats de la comparaison de la sortie $S_i$ avec une base d'apprentissage construite semi-automatiquement

## A.7 Résultats de comparaison de la sortie $R_i$ avec une base d'apprentissage construite semi-automatiquement

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 |
|---|---|---|---|---|---|
| 001 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 002 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 003 | Ok | 18 | 12, 12, 12 | 0 | 11, 11, 11 |
| 004 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 005 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 006 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 007 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 008 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 009 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 010 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 011 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 012 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 013 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 014 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 015 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 017 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 018 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 019 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 020 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 021 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 023 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 024 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 025 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 026 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 |
|---|---|---|---|---|---|
| 027 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 028 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 100 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 101 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 102 | Ok | 19 | 11, 11, 11 | 7 | 6, 0, 6 |
| 103 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 104 | Ok | 19 | 11, 11, 11 | 7 | 2, 8, 8 |
| 105 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 106 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 107 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 108 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 109 | Ok | 19 | 11, 11, 11 | 7 | 6, 0, 6 |
| 110 | Ok | 17 | 11, 11, 11 | 9 | 6, 10, 10 |
| 111 | Ko : Erreur Type 7 | 20 | 12, 12 | 11 | |
| 112 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 113 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 114 | Ok | 19 | 11, 11, 11 | 7 | 6, 0, 6 |
| 115 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 116 | Ok | 19 | 11, 11, 11 | 7 | 2, 8, 8 |
| 117 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 118 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 119 | Ko | Erreurs sur les combinaisons [3, 7] | | | |
| 121 | Ok | 19 | 11, 11, 11 | 7 | 2, 8, 8 |
| 122 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |
| 123 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 124 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 125 | Ok | 18 | 12, 12, 12 | 0 | 11, 11, 11 |
| 128 | Ok | 19 | 11, 11, 11 | 7 | 6, 0, 6 |
| 131 | Ok | 19 | 11, 11, 11 | 7 | 6, 6, 0 |
| 136 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 137 | Ok | 19 | 11, 11, 11 | 7 | 6, 0, 6 |
| 139 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 140 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 141 | Ok | 19 | 11, 11, 11 | 7 | 6, 0, 6 |
| 142 | Ko | Erreurs sur les combinaisons [3, 7] | | | |
| 143 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 144 | Ok | 18 | 12, 12, 12 | 0 | 11, 11, 11 |
| 145 | Ok | 14 | 6, 6, 6 | 0 | 7, 7, 7 |

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 |
|---|---|---|---|---|---|
| 146 | Ok | 19 | 11, 11, 11 | 7 | 0, 6, 6 |
| 147 | Ok | 19 | 11, 11, 11 | 7 | 6, 0, 6 |

TAB. A.7 – Résultats de la comparaison de la sortie $R_i$ avec une base d'apprentissage construite semi-automatiquement

# Annexe B

# Résultats expérimentaux du soustracteur binaire

## B.1 Résultats de comparaison de la sortie $S_i$ du soustracteur avec une base d'apprentissage construite semi-automatiquement

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 |
|---|---|---|---|---|---|
| 029 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 030 | Ko | Erreurs sur les combinaisons [1, 2, 5, 6] et Proche du circuit correct numéro 0, niveau de simplification 2, à une distance de 4 | | | |
| 032 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 033 | Ko : Erreur Type 8 | 13 | Proche du circuit correct numéro 6, niveau de simplification 1, à une distance de 5 | | |
| 034 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 035 | Ok | 0 | 9, 11, 13, 9, 11, 13, 9, 11, 13 | 18, 16, 18, 16, 18, 16 | 27, 27 |
| 036 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 037 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 038 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 039 | Ok | 24 | 17, 17, 19, 17, 17, 19, 11, 13, 19 | 8, 14, 8, 14, 0, 6 | 17, 17 |
| 040 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17,17 |
| 041 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 042 | Ok | 33 | 24, 26, 30, 24, 26, 30, 24, 26, 30 | 17, 21, 17, 21, 17, 21 | 2, 0 |
| 043 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 044 | Ko | Erreurs sur les combinaisons [0, 1, 2, 3, 5, 6] | | | |
| 045 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 046 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 049 | Ok | 0 | 9, 11, 13, 9, 11, 13, 9, 11, 13 | 18, 16, 18, 16, 18, 16 | 27, 27 |
| 051 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 |
|---|---|---|---|---|---|
| 052 | Ko | Erreurs sur les combinaisons [2, 6] et<br>et Proche du circuit correct numéro 4, niveau de simplification 2, à une distance de 5 | | | |
| 053 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 055 | Ko | Erreurs sur les combinaisons [1, 5] et<br>Proche du circuit correct numéro 2, niveau de simplification 2, à une distance de 5 | | | |
| 056 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 057 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 058 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 059 | Ok | 24 | 17, 17, 19, 17, 17, 19, 11, 13, 19 | 8, 14, 8, 14, 0, 6 | 17, 17 |
| 060 | Ko | Erreurs sur les combinaisons [1, 2, 5, 6] | | | |
| 061 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 062 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 064 | Ok | 23 | 16, 12, 18, 16, 16, 18, 16, 16, 18 | 5, 9, 9, 15, 9, 15 | 18, 16 |
| 065 | Ok | 23 | 16, 16, 18, 16, 16, 18, 10, 18, 14 | 11, 11, 11, 11, 5, 0 | 16, 18 |
| 066 | Ko | Erreurs sur les combinaisons [4, 7] et<br>Proche du circuit correct numéro 0, niveau de simplification 2, à une distance de 4 | | | |
| 067 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 069 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 070 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 071 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 073 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 074 | Ko | Erreurs sur les combinaisons [1, 2, 3, 5, 6, 7] | | | |
| 075 | Ko | Erreurs sur les combinaisons [0, 1, 2] et<br>Proche du circuit correct numéro 0, niveau de simplification 2, à une distance de 2 | | | |
| 076 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 077 | Ko : Erreur Type 2 | 13 | | | |
| 078 | Ko | Erreurs sur les combinaisons [2, 4] | | | |
| 079 | Ko : Erreur Type 2 | 9 | Proche du circuit correct numéro 1, niveau de simplification 2, à une distance de 4 | | |
| 080 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 083 | Ko : Erreur Type 7 | 21 | 11, 11, 13 | 9 | |
| | | Proche du circuit numéro 1, niveau de simplification 1, à une distance de 1 | | | |
| 084 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 085 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 086 | Ok | 24 | 11, 13, 19, 17, 17, 19, 17, 17, 19 | 0, 6, 8, 14, 8, 14 | 17, 17 |
| 087 | Ko : Erreur Type 6 | 31 | 23, 25, 23 | 19 | |

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 |
|---|---|---|---|---|---|
| 088 | Ko : Erreur Type 8 | 13 | Proche du circuit correct numéro 0, niveau de simplification 1, à une distance de 5 | | |
| 089 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 090 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 091 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 092 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 093 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 094 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 095 | Ko | Erreurs sur les combinaisons [4, 5, 6, 7] et Proche du circuit correct numéro 0, niveau de simplification 3, à une distance de 2 | | | |
| 096 | Ok | 33 | 24, 28, 30, 24, 28, 30, 24, 26, 28 | 19, 19, 19, 19, 17, 19 | 0, 8 |
| 097 | Ok | 15 | 12, 0, 10, 10, 10, 12, 10, 10, 12 | 11, 15, 15, 15, 15, 15 | 26, 24 |
| 098 | Ok | 33 | 24, 26, 28, 24, 28, 30, 24, 28, 30 | 17, 19, 19, 19, 19, 19 | 0, 8 |
| 099 | Ok | 33 | 24, 28, 28, 24, 28, 28, 24, 28, 28 | 17, 19, 17, 19, 17, 19 | 0, 2 |
| 159 | Ko | Erreurs sur les combinaisons [2, 3, 6, 7] | | | |

TAB. B.1 – Résultats de la comparaison de la sortie $S_i$ du soustracteur avec une base d'apprentissage construite semi-automatiquement

## B.2 Résultats de comparaison de la sortie $R_i$ du soustracteur avec une base d'apprentissage construite semi-automatiquement

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 | Niv 4 |
|---|---|---|---|---|---|---|
| 029 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 030 | Ko | Erreurs sur les combinaisons [2, 3, 4, 5] | | | | |
| 032 | Ko | Erreurs sur les combinaisons [4, 7] | | | | |
| 033 | Ok | 21 | 11, 11, 13 | 9 | 8, 8 | |
| 034 | Ko : Erreur Type 8 | * | 14, 14 | 13 | | |
| 035 | Ko | Erreurs sur les combinaisons [4, 5, 7] et Proche du circuit correct numéro 0, niveau de simplification 3, à une distance de 4 | | | | |
| 036 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 037 | Ko : Erreur Type 8 | * | 14, 14 | 13 | | |
| 038 | Ok | 19 | 9, 9, 13 | 7 | 6, 0 | |
| 039 | Ok | 21 | 11, 13, 13 | 9 | 4, 8 | |
| 040 | Ko : Erreur Type 7 | 11 | Proche du circuit correct numéro 0, niveau de simplification 3, à une distance de 3 | | | |
| 041 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 042 | Ok | 18 | 8, 8, 10 | 0 | 7, 9 | |

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 | Niv 4 |
|---|---|---|---|---|---|---|
| 043 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 044 | Ko | Erreurs sur les combinaisons [2, 3] | | | | |
| 045 | Ok | 18 | 8, 8, 10 | 0 | 7, 9 | |
| 046 | Ko | Erreurs sur les combinaisons [2, 6] | | | | |
| 049 | Ko | Erreurs sur les combinaisons [4, 5, 7] et<br>Proche du circuit correct numéro 0, niveau de simplification 3, à une distance de 4 | | | | |
| 051 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 052 | Ok | 21 | 11, 11, 13 | 9 | 8, 8 | |
| 053 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 055 | Ko | Erreurs sur les combinaisons [4, 5, 7] et<br>Proche du circuit correct numéro 0, niveau de simplification 3, à une distance de 2 | | | | |
| 056 | Ko | Erreurs sur les combinaisons [3, 7] et<br>Proche du circuit correct numéro 0, niveau de simplification 3, à une distance de 5 | | | | |
| 057 | Ko | Erreurs sur les combinaisons [6, 7] | | | | |
| 058 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 059 | Ok | 21 | 11, 11, 13 | 9 | 8, 8 | |
| 060 | Ok | 21 | 11, 11, 13 | 9 | 8, 10 | |
| 061 | Ko | Erreurs sur les combinaisons [4, 6, 7] | | | | |
| 062 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 064 | Ko | Erreurs sur les combinaisons [5, 6] | | | | |
| 065 | Ok | 21 | 11, 11, 13 | 9 | 8, 8 | |
| 066 | Ok | 15 | 9, 5, 7 | 7 | 12 | |
| 067 | Ok | 19 | 9, 9, 13 | 7 | 6, 0 | |
| 069 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 070 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 071 | Ko | Erreurs sur les combinaisons [4, 6] | | | | |
| 073 | Ok | 21 | 11, 13, 13 | 9 | 4, 8 | |
| 074 | Ko | Erreurs sur les combinaisons [5, 7] | | | | |
| 075 | Ko | Erreurs sur les combinaisons [4, 5, 6, 7] | | | | |
| 076 | Ko | Erreurs sur les combinaisons [3, 5] | | | | |
| 077 | Ko | Erreurs sur les combinaisons [2, 3, 4] | | | | |
| 078 | Ko : Erreur Type 6 | 21 | 11 | 7, 9, 5, 7 | 16 | 15 |
| 079 | Ko | Erreurs sur les combinaisons [4, 5, 7] et<br>Proche du circuit correct numéro 0, niveau de simplification 3, à une distance de 5 | | | | |
| 080 | Ko | Erreurs sur les combinaisons [0, 3] | | | | |
| 083 | Ok | 21 | 11, 11, 13 | 9 | 8, 8 | |
| 084 | Ok | 19 | 9, 9, 13 | 7 | 6, 0 | |
| 085 | Ok | 18 | 8, 8, 10 | 0 | 7, 9 | |

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 | Niv 4 |
|---|---|---|---|---|---|---|
| 086 | Ok | 19 | 9, 9, 13 | 7 | 6, 0 | |
| 087 | Ko : Erreur Type : 6 | '* ' | 10 | 6, 4, 4, 0 | 13 | 12 |
| | | Proche du circuit correct numéro 0, niveau de simplification 1, à une distance de 3 | | | | |
| 088 | Ko : Erreur Type : 8 | 17 | 9, 9 | 8 | | |
| 089 | Ko | Erreurs sur les combinaisons [2, 7] | | | | |
| 090 | Ok | 21 | 11, 11, 13 | 9 | 8, 8 | |
| 091 | Ok | 21 | 11, 11, 13 | 9 | 8, 8 | |
| 092 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 093 | Ko : Erreur Type : 3 | 20 | 12, 12 | 11 | | |
| 094 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 095 | Ko | Erreurs sur les combinaisons [1, 6] | | | | |
| 096 | Ok | 21 | 11, 11, 13 | 9 | 8, 8 | |
| 097 | Ko | Erreurs sur les combinaisons [4, 6] | | | | |
| 098 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 099 | Ok | 21 | 11, 11, 13 | 7 | 0, 6 | |
| 159 | Ok | 21 | 11, 11, 13 | 9 | 8, 8 | |

TAB. B.2 – Résultats de la comparaison de la sortie $R_i$ du soustracteur avec une base d'apprentissage construite semi-automatiquement

# Annexe C

# Résultats expérimentaux du complément à 2

## C.1 Résultats de comparaison de la sortie $S_1$ du complément à 2 avec une base d'apprentissage construite semi-automatiquement

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 | Niv 4 | Niv 5 | Niv 6 | Niv 7 |
|---|---|---|---|---|---|---|---|---|---|
| 080 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 081 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 083 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 084 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 087 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 095 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 150 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 162 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 165 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 168 | Ko : Erreur Type : 13 | 76 | 65 | 43, 43, 43 | 32, 32, 32, 30, 32, 32, 32, 32, 32, 32, 32, 32, 30, 32, 32, 30, 32 | 24, 22, 24, 22, 20, 20 | 14, 14, 14 | 3 | 0 |

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 | Niv 4 | Niv 5 | Niv 6 | Niv 7 |
|---|---|---|---|---|---|---|---|---|---|
| 169 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 173 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 181 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 194 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 201 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |
| 203 | Ok | 78 | 45, 45, 45 | 33, 33, 33, 34, 34, 34, 34, 34, 34 | 14, 14, 14 | 0 | | | |

TAB. C.1 – Résultats de la comparaison de la sortie $S_1$ du complément à 2 avec une base d'apprentissage construite semi-automatiquement

## C.2 Résultats de comparaison de la sortie $S_2$ du complément à 2 avec une base d'apprentissage construite semi-automatiquement

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 | Niv 4 | Niv 5 | Niv 6 |
|---|---|---|---|---|---|---|---|---|
| 080 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 081 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 083 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 084 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 087 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 095 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 150 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 162 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 165 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 168 | Ko : Type Erreur 8 | * | *, * | *, *, *, *, *, *, *, *, *, *, * | *, *, *, * | * | 13 | 12 |
| 169 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 173 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 | Niv 4 | Niv 5 | Niv 6 |
|---|---|---|---|---|---|---|---|---|
| 181 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 194 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 201 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |
| 203 | Ok | 86 | 37, 37, 38, 38 | 16 | 0 | | | |

TAB. C.2 – Résultats de la comparaison de la sortie $S_2$ du complément à 2 avec une base d'apprentissage construite semi-automatiquement

## C.3 Résultats de comparaison de la sortie $S_3$ du complément à 2 avec une base d'apprentissage construite semi-automatiquement

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 | Niv 4 | Niv 5 | Niv 6 |
|---|---|---|---|---|---|---|---|---|
| 080 | Ok | * | *,* | * | 17, *, * | 7, 7 | 0 | 22 |
| 081 | Ok | * | *,* | * | *,*,* | *,* | * | 0 |
| 083 | Ok | * | *,* | * | *,*,* | 13, 13 | 10 | 18 |
| 084 | Ok | * | *,* | * | *,*,* | *,* | * | 0 |
| 087 | Ok | * | *,* | * | 17, *, * | 7,7 | 0 | 22 |
| 095 | Ko | Erreurs sur les combinaisons [4, 12] | | | | | | |
| 150 | Ok | * | *,* | * | *,*,* | 13, 13 | 10 | 18 |
| 162 | Ko | Erreurs sur les combinaisons [4, 12] | | | | | | |
| 165 | Ko | Erreurs sur les combinaisons [0, 2, 8, 10] | | | | | | |
| 168 | Ok | * | *,* | * | *, *, * | *,* | * | * |
| 169 | Ok | * | *,* | * | 22, 23, 23 | 12,12 | 13 | * |
| 173 | Ok | * | *,* | * | 20,*,* | 10,10 | 9 | 19 |
| 181 | Ok | * | *,* | * | *,*,* | *,* | * | 0 |
| 194 | Ok | * | *,* | * | *,*,* | *,* | * | 0 |
| 201 | Ok | * | *,* | * | *,*,* | 13,13 | 10 | 18 |
| 203 | Ok | * | *,* | * | *,*,* | 13,13 | 10 | 18 |

TAB. C.3 – Résultats de la comparaison de la sortie $S_3$ du complément à 2 avec une base d'apprentissage construite semi-automatiquement

## C.4 Résultats de comparaison de la sortie $S_4$ du complément à 2 avec une base d'apprentissage construite semi-automatiquement

| Circuit | Statut | Niv 0 | Niv 1 | Niv 2 | Niv 3 | Niv 4 |
|---|---|---|---|---|---|---|
| 080 | Ok | * | *, *, *, *, *, *, *, * | 17, 17, 17, 17, 17, 17 | 0 | * |
| 081 | Ok | * | *, *, *, *, *, *, *, * | *, *, *, *, *, * | * | 0 |
| 083 | Ok | * | *, *, *, *, *, *, *, * | *, *, *, *, *, * | * | * |
| 084 | Ok | * | *, *, *, *, *, *, *, * | *, *, *, *, *, * | * | 0 |
| 087 | Ok | * | *, *, *, *, *, *, *, * | 17, 17, 17, 17, 17, 17 | 0 | * |
| 095 | Ko : Erreur Type 3 | * | *, *, * | 20, 20, 20 | 10, 10 | 9 |
| 150 | Ok | * | *, *, *, *, *, *, *, * | *, *, *, *, *, * | 16 | * |
| 162 | Ko : Erreur Type 5 | * | *, *, * | 20, 20, 20 | 10, 10 | 9 |
| 165 | Ok | * | *, *, *, *, *, *, *, * | *, *, *, *, *, * | * | 0 |
| 168 | Ok | * | *, *, *, *, *, *, *, * | *, *, *, *, *, * | * | 0 |
| 169 | Ko : Erreur Type 9 | * | *, *, *, *, *, *, *, * | *, *, *, *, *, * | 20 | |
| 173 | Ok | * | *, *, *, *, *, *, *, * | *, *, *, *, *, * | 16 | * |
| 181 | Ko : Erreur Type 9 | * | *, *, *, *, *, *, *, * | *, *, *, *, *, * | 23 | |
| | | Proche du circuit correct 0, niveau de simplification 4, à une distance de 1 | | | | |
| 194 | Ko | Erreurs sur les combinaisons [2, 4, 6, 8] | | | | |
| 201 | Ok | * | | | | |
| 203 | Ko | Erreurs sur les combinaisons [8, 11] | | | | |

TAB. C.4 – Résultats de la comparaison de la sortie $S_4$ du complément à 2 avec une base d'apprentissage construite semi-automatiquement

# Annexe D

# Types Abstraits de Données

## D.1 Expression Booléenne ou EBD

Le type abstrait de données « EBD » est une formalisation des expressions booléennes développées en « première forme canonique : somme des termes produit ». Elle est représentée sous format d'une liste de termes booléens. Chaque terme (le TAD correspondant est « TermeEB ») fait apparaitre une représentation de toutes les entrées du circuit logique correspondant. Le TAD correspondant est le suivant :

| | | |
|---|---|---|
| **Nom**: | EBD | |
| **Utilise**: | SE, TermeEB, TdV, **Chaine**, **Naturel**, **Booleen**, Ensemble | |
| **Opérations**: | créerEBD: | TdV × **Chaine** → EBD |
| | obtenirTdVEBD: | EBD → TdV |
| | obtenirNombreDeTermesEBD: | EBD → **Naturel** |
| | obtenirTermeEBD: | EBD × **Naturel** → TermeEB |
| | obtenirTermesAssociésEBD: | EBD × TermeEB → Ensemble<TermeEB> |
| | associerTermesEBD: | EBD × TermeEB × Ensemble<TermeEB> → Ensemble<EBD> |
| | obtenirStatutEBD: | EBD → **Chaine** |
| | obtenirNiveauSimplificationEBD: | EBD → **Chaine** |
| | ajouterEtiquetteEDB: | EBD × **Chaine** → EBD |
| | ajouterEtiquetteEDB: | EBD × **Chaine** → EBD |
| | obtenirEtiquetteEDB: | EBD → **Chaine** |
| | obtenirMcCluskyEBD: | EBD → EBD |

| | | |
|---|---|---|
| | obtenirSE_EBD: | EBD → SE |
| **Sémantiques**: | créerEBD: | construit une expression booléenne à partir d'une table de vérité et un statut, correct ou incorrect. |
| | obtenirTdVEBD: | retourne une table de vérité à partir de l'expression booléenne. |
| | obtenirNombreTermesEBD: | retourne le nombre de termes « produit » de l'expression booléenne. |
| | obtenirTermeEBD: | retourne un terme de l'expression booléenne. |
| | obtenirTermesAssociésEBD: | retourne tous les termes qu'on peut associer à un terme donné de l'expression booléenne pour effectuer une simplification. |
| | associerTermesEBD: | retourne les expressions booléennes simplifiées pour un terme donné. |
| | obtenirStatutEBD: | retourne « Ok », si l'expression booléenne est correcte, « Ko + Type d'erreur » dans le cas contraire. |
| | obtenirNiveauSimplificationEBD: | retourne le niveau de simplification de l'expression booléenne. |
| | ajouterEtiquetteEDB: | affecte une « étiquette » à l'expression booléenne. |
| | obtenirEtiquetteEDB: | retourne l'étiquette correspondant à l'expression booléenne. |
| | obtenirMcCluskyEBD: | retourne l'expression simplifiée en utilisant la méthode de McClusky. |
| | obtenirSE_EBD: | retourne le schéma électronique correspondant à l'expression booléenne. |

## D.2 Terme Booléen

Le type abstrait de données « TermeEB » est une chaîne composée des binaires « 0 » et « 1 », et de « X ». Le « X » indique que l'entrée correspondante a été simplifiée pour ce terme. Le TAD correspondant est le suivant :

| | | |
|---|---|---|
| **Nom**: | TermeEB | |
| **Utilise**: | TdV, TypeBinaire=$\{0, 1, X\}$, **Naturel**, **Booleen** | |
| **Opérations**: | créerTermeEB: | TdV $\times$ **Naturel** $\rightarrow$ TermeEB |
| | associableTermesEB: | TermeEB $\times$ TermeEB $\times$ **Naturel** $\rightarrow$ **Booleen** |
| | associerTermesEB: | TermeEB $\times$ TermeEB $\rightarrow$ TermeEB |
| | obtenirNombreDeBitsTermeEB: | TermeEB $\rightarrow$ **Naturel** |
| | obtenirBitTermeEB: | TermeEB $\times$ **Naturel** $\rightarrow$ TypeBinaire |
| **Sémantiques**: | créerTermeEB: | crée un terme à partir d'une entrée de la table de vérité. |
| | associableTermesEB: | indique si deux termes peuvent être combinés pour une simplification. |
| | associerTermesEB: | combine deux termes pour en extraire un autre terme simplifié. |
| | obtenirNombreDeBitsTermeEB: | retourne le nombre de bits constituant un terme. |
| | obtenirBitTermeEB: | retourne un bit d'un terme. |

# Bibliographie

# Bibliographie


Abernot, Y. (2002). *La pédagogie : une encyclopédie pour aujourd'hui*, chapitre L'évaluation scolaire, pages 235–245. ESF (sous la direction de Jean HOUSSAYE).

Akers, S. (1978). Binary decision diagrams. *In IEEE Transactions on Computers*, volume 27, pages 509–516.

Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., Raths, J. et Wittrock, M. C. (2001). *A Taxonomy for Learning, Teaching, and Assessing - A Revision of Bloom's Taxonomy of Educational Objectives*. Addison Wesley Longman.

Astolfi, J.-P. (2001). *L'erreur, un outil pour enseigner*. ESF, 4 édition.

Auxepaules, L. (2009). *Analyse des diagrammes de l'apprenant dans un EIAH de la modélisation orientée objet*. Thèse de doctorat, Université du Maine. pages 226.

Barbier, J.-M. (2001). *L'évaluation en formation*. Puf Education et Formation, 5e édition. pages 310.

Baron, M. (1994). Eiao : quelques repères. *Revue Terminal*, 65:67–84.

Benomar, K. (2007). Développement d'un logiciel d'interprétation de schémas électroniques. Mémoire d'ingénieur, INSA de Rouen. pages 70.

Bloom, B. (1956). *Taxonmy of educational objectives : The classification of educational goals. Handbook I, cognitive domain*. Longman, New York.

Boole, G. (1854). *An Investigation into the Laws of Though on Which Are Founded the Mathematical Theories of Logic and Probabilities*. Dover Publications, New York.

Bruillard, E. (1997). *Les machines à enseigner (Chapitre 5 : Des tuteurs intelligents aux environnements interactifs)*. Hermès, Paris. pages 153–196.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691.

Buche, C. (2005). *Un système tutoriel intelligent et adaptatif pour l'apprentissage de compétences en environnement virtuel de formation*. Thèse de doctorat, Université de Bretagne Occidentale. pages 246.

Bunke, H. (1997). On a relation between graph edit distance and maximum common subgraph. *In Pattern Recognition Letters*, volume 18, pages 689–694. Elsevier.

Bunke, H. (1998). Error-tolerant graph matching : A formal framework and algorithms. *In Advances in Pattern Recognition*, volume 1451, pages 1–14. Springer.

Bunke, H. (2000). Graph matching : Theoretical foundations, algorithms, and applications. *In Proc. Vision Interface 2000, Montreal*, pages 82–88.

Bunke, H. et Allerman, G. (1983). Inexact graph matching for structural pattern recognition. *In Pattern Recongnition Letters*, volume 1, pages 245–253. Elsevier Science Publisher.

Bunke, H. et Messmer, B. (1997). Recent advances in graph matching. *In International Journal of Pattern Recognition and Artificial Intelligence*, volume 11, pages 169–203. World Scientific.

Bunke, H. et Shearer, K. (1998). A graph distance metric based on the maximal common subgraph. *In Pattern recognition letters*, volume 19, pages 255–259.

Burch, C. (2002). Logisim : A graphical system for logic circuit design and simulation. *Journal of Educational and Resources in Computing*, 2:5–16.

Cardinet, J. (1988). *Evaluation scolaire et mesure*. De Boeck, 2 édition.

Conati, C., Gertner, A., Vanlehn, K. et Druzdzel, M. (1997). On-line student modeling for coached problem solving using bayesian networks. *In User Modeling : Proceedings of the Sixth International Conference*, pages 231–242. Springer.

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

Corneil, D. et Gotlieb, C. (1970). An efficient algorithm for graph isomorphism. *In Journal of the Association for Computing Machinery*, volume 17, pages 51–64.

Cuissart, B. (2004). *Plus grande structure commune à deux graphes : méthode de calcul et intérêt dans un contexte SAR*. Thèse de doctorat, Université de CAEN. pages 152.

Darche, P. (2002). *Architectures des ordinateurs : Fonctions booléennes, logiques combinatoire et séquentielle*, volume 2. Vuibert Informatique. pages 416.

Delorme, F. (2005). *Evaluation et modélisations automatiques des connaissances des apprenants à l'aide de cartes conceptuelles*. Thèse de doctorat, INSA de Rouen. pages 182.

Denis, F. et Gilleron, R. (2000). *Apprentissage à partir d'exemples - Notes de cours*. Université Lille3. (http ://www.grappa.univ-lille3.fr/polys/apprentissage/, dernière visite : juin 2009).

Dreyfus, G., Martinez, J.-M., Samuelides, M., Gordon, M., Badran, F., Thiria, S. et Hérault, L. (2004). *Réseaux de neurones*. Eyrolles. pages 386.

Duda, R., Hart, P. et Stork, D. (2001). *Pattern Classification*. Wiley Interscience. pages 738.

Gagne, R. M. (1965). *The conditions of learning*. New York, Holt, Rinehart & Winston.

Genest, D. (2000). *Extension du modèle des graphes conceptuels pour la recherche d'informations*. Thèse de doctorat, Université de Montpellier II.

Gilleron, R. et Tommasi, M. (2000). *Découverte de connaissances à partir de données - Notes de cours*. Université Lille3. (http ://www.grappa.univ-lille3.fr, dernière visite : juin 2009).

Guttag, J. V. (1975). *The specification and application to programming of abstract data types*. Thèse de doctorat, University of Toronto - Canada.

Hadji, C. (1999). *L'évaluation démystifiée*. ESF, 2 édition. pages 126.

Hadji, C. (2000). *L'évaluation, règles du jeu : Des intentions à l'action*. ESF, 6 édition. pages 191.

Hart, P. E., Nilsson, N. J. et Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *In IEEE Transactions on Systems Science and Cybernetics*, volume SSC-4 de *2*, pages 100–107.

Jean-Daubias, S. (2000). *PEPITE : un système d'assistance au diagnostic de compétences*. Thèse de doctorat, Université du Maine. pages 293.

Jiang, X., Münger, A. et Bunke, H. (2000). Synthesis of representative graphical symbols by computing generalized median graph. *In GREC'99, LNCS 1941*, pages 183–192. Springer-Verlag.

Jorro, A. (2000). *L'enseignant et l'évaluation : Des gestes évaluatifs en question*. De Boeck Université. pages 184.

Karnaugh, M. (1953). The map method for synthesis of combinational logic circuits. *In Trans. AIEE*, volume 72, pages 593–599.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69.

Labat, J.-M. (2002). Eiah : Quel retour d'informations pour le tuteur ? *In TICE 2002*, pages 81–88.

Landsheere, D. (1984). *Définir les objectifs de l'éducation*. PUF. pages 340.

Lemarchand, S., Degrugillier, D., Galisson, A. et Gimenes, C. (2000). L'électronique en questions : premier titre de la collection hypermédia pédagogique du groupe des ecoles des télécommunications. *In TICE 2000*, pages 165–166.

Lepage, P. et Romainville, M. (2009). Le questionnaire à choix multipe n°69 - réseaux : Revue au service de l'enseignement et de l'apprentissage à l'université. (http ://www.fundp.ac.be/recherche/publications/page_view/67449/).

Levi, G. (1972). A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *In Calcolo*, numéro 9, pages 341–354.

Lieury, A. (1996). *Manuel de psychologie de l'éducation et de la formation*. Dunod. pages 415.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *In Proceedings of the Fifth Berkeley Symposium on Mathematical statistics and probability*, volume 1, pages 281–297. Berkeley, University of California Press.

Maulini, O. (1996). Qui a eu cette idée folle un jour d'inventer les notes à l'école ? petite histoire de l'évaluation chiffrée à l'usage de celles et ceux qui désirent s'en passer. (http ://www.unige.ch/fapse/SSE/teachers/maulini/note.html, dernière visite : juillet 2007).

McCluskey, E. (1956). Minimization of boolean functions. *In Bell System Tech. J.*, volume 35, pages 1417–44.

McGregor, J. (1982). Backtrack search algorithms and the maximal common subgraph problem. *In Software-Practice and Experience*, numéro 12, pages 23–34.

Meirieu, P. (2002). *Apprendre... oui, mais comment*. ESF, 18 édition. pages 192.

Mendelsohn, P. et Dillenbourg, P. (1993). *Le développement de l'enseignement intelligemment assisté par ordinateur*. PUF.

Messmer, B. T. et Bunke, H. (1998). A new algorithm for error-tolerant subgraph isomorphism detection. *In IEEE Transaction on Pattern Analysis and Machine Intelligence*, volume 20, pages 493–504.

Meyer, B. (2000). *Conception et programmation orientées objet*. Eyrolles. pages 1224.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Science. pages 421.

Naïm, P., Wuillemin, P.-H., Leray, P., Pourret, O. et Becker, A. (2004). *Réseaux Bayésiens*. Eyrolles. pages 424.

Neuhaus, M. et Bunke, H. (2005). A graph matching based approach to fingerprint classification using directional variance. *In AVBPA 2005, LNCS 3546*, pages 191–200. Springer-Verlag.

Neuhaus, M., Riesen, K. et Bunke, H. (2006). Fast suboptimal algorithms for the computation of graph edit distance. *In 11th International Workshop on Structural and Syntactic Pattern Recognition*, pages 163–172. Springer.

Nicaud, J. (1987). *Aplusix : un système expert en résolution pédagogique d'exercices d'algèbre*. Thèse de doctorat, Université Paris XI-Orsay. pages 189.

Nicaud, J., Bittar, M., Chaachoua, H., Inamdar, P. et Maffei, L. (2007). Experiments of aplusix in four countries. *International Journal for Technology in Mathematics Education*, 13(2).

Nilsson, N. (1980). *Principles of Artificial Intelligence*. Tioga Publishing.

Nketsa, A. (1998). *Circuits logiques programmables*. Ellipses. pages 1–23 (256).

Novak, J. D. et Gowin, D. B. (1984). *Learning how to learn*. Cambridge University Press. pages 200.

Pearce, A., Caelli, T. et Bischof, W. F. (1994). Rulegraphs for graph matching in pattern recognition. *In Pattern Recognition*, volume 27, pages 1231–1247.

Peretti, A. D., Boniface, J. et Legrand, J.-A. (2000). *Encyclopédie de l'évaluation en formation et en éducation : Guide pratique*. ESF, 2 édition.

Perrenoud, P. (1999). *L'évaluation des élèves : De la fabrication de l'excellence à la régulation des apprentissages. Entre deux logiques.* De Boeck Université. pages 219.

Perrenoud, P. (2000). *Pédagogie Différenciée : des intentions à l'action*. ESF, 2 édition. pages 194.

Preux, P. (2008). *Fouille de données - Notes de cours.* Université Lille3. (http ://www.grappa.univ-lille3.fr/ ppreux/fouille, dernière visite : juin 2009).

Prévit, D. (2008). *Génération d'exercices et analyse multicritère automatique de réponses ouvertes - PépiGen, un système auteur en algèbre élémentaire*. Thèse de doctorat, Université du Maine.

Pudelko, B., Basque, J. et Legros, D. (2003). Vers une méthode d'évaluation des cartes conceptuelles fondée sur l'analyse en systèmes. *In EIAH*, pages 555–558.

Quine, W. (1952). The problem of simplifying truth functions. *In Amer. Math. Monthly*, volume 59, pages 521–31.

Riesen, K. et Bunke, H. (2008). Approximate graph edit distance computation by means of bipartite graph matching. *Elsevier*, 27(7):950–959.

Rodríguez-Estévez, J., Caeiro-Rodríguez, M. et Santos-Gago, J. M. (2003). Standardization in computer based learning. *Upgrade*, IV(5).

Rosenblatt, F. (1958). A probalistic model for information storage and organisation in the brain. *In Psycological Review*, volume 65, pages 386–408.

Rumelhart, D. E. et McClelland, J. L. (1986). *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*. MIT Press, Cambridge. pages 567.

Saux, B. L. et Bunke, H. (2000). Feature selection for graph-based image classifiers. *In IbPRIA 2005, LNCS 3523*, pages 147–154. Springer-Verlag.

Scriven, M. (1967). the methodology of evaluation. *In A.E.R.A. Monograph Series on Curriculum Evaluation*, pages 39–83. Rand. Mc Nally.

Serrell, R. (1953). Elements of boolean algebra for the study of information-handling systems. *In Proceeding of IRE*, volume 41, pages 1366–1380.

Shannon, C. (1938). A symbolic analysis of relay and switching circuits. *In Transactions of the American Institute of Electrical Engineers*, volume 57, pages 713–723.

Sorlin, S. (2006). *Mesurer la similarité de graphes*. Thèse de doctorat, Université Claude Bernard Lyon I. pages 154.

Suard, F. (2006). *Méthodes à noyaux pour la détection de piétons*. Thèse de doctorat, INSA de Rouen. pages 131.

Tanana, M., Delestre, N., Pécuchet, J.-P. et Bennouna, M. (2008a). Plate-forme web pour la gestion et l'evaluation de travaux pratiques en electronique numérique. *In Colloque International Organisation Numérique des Universités (CIONU 2008)*.

Tanana, M., Delestre, N., Pécuchet, J.-P. et Bennouna, M. (2008b). Évaluation du savoir-faire en électronique numérique à l'aide d'un algorithme de classification. *In Colloque International TICE 2008*, pages 44–51.

Tanana, M., Delestre, N., Pécuchet, J.-P. et Bennouna, M. (2009). Génération d'exemples pour l'évaluation de l'apprenant en électronique numérique à l'aide d'un algorithme de classification. *In Conférence EIAH 2009*, pages 345–352.

Tanana, M., Pécuchet, J.-P. et Guégot, F. (2006). Un outil pour l'évaluation automatique des apprenants. *In Colloque International TICE 2006*, pages 2.

Tchounikine, P. (2009). Précis de recherche en ingénierie des eiah. (http ://membres-liglab.imag.fr/tchounikine/Precis.html, dernière viste : juillet 2009).

Tribollet, B., Langlois, F. et Jacquet, L. (2000). Protocoles d'emploi des cartes conceptuelles au lycée et en formation des maîtres. *In TREMA*, volume 18. IUFM de Montpellier,.

Tsai, W. et Fu, K. (1979). Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *In IEEE Translation on Systems Man and Cybernet*, volume 9, pages 757–768.

Turton, B. C. H. (1996). Extending quine-mccluskey for exclusive-or logic synthesis. *In IEEE Transaction on Education*, volume 39, pages 81–85.

Ullmann, J. (1976). An algorithm for subgraph isomorphism. *In Journal of the Assocmtlon for Computing Machinery*, volume 23, pages 31–42.

Unger, S. H. (1964). Git-a heuristic program for testing pairs of directed line graphs for isomorphism. *Communications of the ACM*, 7(1):26–34.

Vanlehn, K., Lynch, C., Schulze, K., Shapiro, J., Shelby, R., Taylor, L., Tracy, D., Weinstein, A. et Wintersgill, M. (2005). The andes physics tutoring system : Five years of evaluations. *In Proceedings of the Artificial Intelligence in Education Conference*.

Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley. pages 768.

Vollmer, H. (1999). *Introduction to Circuit Complexity : a Uniform Approach*. Springer-Verlag. pages 270.

Wong, A. K. C., You, M. et Chan, S. C. (1990). An algorithm for graph optimal monomorphism. *In IEEE Transactions on Systems, MAN and Cybernetics*, volume 20, pages 628–636.

Zanella, P. et Ligier, Y. (1998). *Architecture et Technologie des ordinateurs*. Dunod, 3° édition. pages 71–87 (495).