HAL
open science

Génération automatique de sujets d'évaluation individuels en contexte universitaire

Richardson Ciguene

To cite this version:

Richardson Ciguene. Génération automatique de sujets d'évaluation individuels en contexte universitaire. Autre [cs.OH]. Université de Picardie Jules Verne, 2019. Français. NNT : 2019AMIE0046 . tel-03638132

HAL Id: tel-03638132

https://theses.hal.science/tel-03638132v1

Submitted on 12 Apr 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse de Doctorat

Mention informatique

présentée à ***l'Ecole Doctorale en Sciences Technologie et Santé (ED 585)***

de l'Université de Picardie Jules Verne

Richardson CIGUENE

pour obtenir le grade de Docteur de l'Université de Picardie Jules Verne dans le cadre d'une cotutelle avec l'Ecole Supérieure d'Infotronique d'Haïti

Génération automatique de sujets d'évaluation individuels en contexte universitaire

Soutenue le 14/11/2019, après avis des rapporteurs, devant le jury d'examen :

M. E. EMMANUEL, Professeur, Université Quisqueya Haïti	**Président**
M. D. LENNE, Professeur, Université de Technologie de Compiègne	**Rapporteur**
M. S. GARLATTI, Professeur, IMT Atlantique Bretagne	**Rapporteur**
Mme L. B. DEVENDEVILLE, Maître de Conférences, UPJV	**Examinateur**
Mme D. GROUX-LECLET, Maître de Conférences HDR, UPJV	**Examinateur**
M. G. DEQUEN, Professeur, UPJV	**Directeur**
M. B. M. TOUSSAINT, Docteur, ESIH Haïti	**Co-directeur**
Mme C. JOIRON, Maître de Conférences, UPJV	**Encadrant**
M. P. ATTIE, Enseignant, ESIH Haïti	**Invité**

Je dédie ce travail
à ma grand-mère, Mme Ophane CIGUENE

À mes directeurs bien-aimés !

À mon épouse et ma fille !

Remerciements

Je remercie grandement toutes les personnes qui m'ont accompagné durant ces années de recherche, plus spécifiquement :

- M. Gilles Dequen, Professeur à l'Université de Picardie Jules Vernes, pour m'avoir accepté en thèse et m'avoir encadré du début à la fin. Je remercie la confiance qu'il a placée en moi tout au long de ces années de recherches, grâce à laquelle je peux aboutir, aujourd'hui, à ces résultats.
- M. Dominique Lenne, Professeur à l'UTC, qui me fait l'honneur d'être rapporteur de cette thèse tout en ayant grandement contribué à l'édification de ce travail par ses conseils et recommandations.
- M. Serge Garlatti, Professeur à l'IMT Atlantique, qui me fait l'honneur d'être rapporteur de cette thèse. Ses conseils m'ont été très précieux.
- Mme Dominique Groux-Leclet, Maître de Conférence à l'Université de Picardie Jules Verne, d'avoir accepté d'examiner ces travaux de recherche, pour ses suggestions et aussi pour avoir accepté de participer au jury.
- Mme Laure Brisoux, Maître de Conférence à l'Université de Picardie Jules Verne, d'avoir accepté d'examiner ces travaux de recherche, pour ses conseils et d'avoir accepté de participer au jury.
- M. Evens Emmanuel, Professeur à l'UNIQ d'Haiti, d'avoir accepté d'examiner ces travaux de recherche, pour ses suggestions et avoir accepté de présider le jury
- M. Ben-Manson Toussaint, Docteur à l'Ecole Supérieure d'Infotronique d'Haïti, pour avoir co-encadré mes dernières années de thèse. Sa disponibilité, son sens d'écoute et ses conseils ont aidé à la réalisation de cette thèse.
- M. Patrick Attie, Directeur-Enseignant à l'Ecole Supérieure d'Infotronique d'Haïti pour ses conseils d'avoir accepté de rejoindre le jury.
- M. Kenny Robert Philippe Auguste, Docteur à l'Ecole Supérieure d'Infotronique d'Haïti pour avoir co-encadré mes premières années de thèse. Ses conseils avisés et sa motivation ont grandement contribué à l'aboutissement de ces travaux.

Je suis très particulièrement reconnaissant à Mme Céline Joiron, Maître de Conférence à l'Université de Picardie Jules Verne, d'avoir très rigoureusement encadré ce travail de recherche. Sans sa motivation, son écoute, sa compréhension, sa patience, ses compétences scientifiques, la réalisation de cette thèse ne serait pas possible. Je dois très grandement tout ce dont j'ai pu apprendre tout au long de cette thèse à elle. Merci !»

Remerciements spéciaux à mes collègues des laboratoires MIS et SITERE pour avoir contribué à ma motivation quotidienne ; merci aux institutions telles-que l'AUF, le CDH et l'ESIH pour leurs supports administratifs et financiers ;

Aussi, je remercie les membres de ma famille qui depuis les Antilles m'ont toujours soutenu ; ma très chère épouse Saëlle et ma fille adorée Thaëlle pour leur encouragement

au quotidien.

Je remercie Dieu qui m'accompagne partout et dans tout ce que j'entreprends.

Sommaire

Résumé **ix**

Remerciements **xi**

Sommaire **xiii**

Introduction générale **1**

1 Contexte pédagogique **5**

2 Contexte scientifique et objectifs **23**

3 Une métrique pour la différenciation dans les évaluations **33**

4 Génération de sujets différenciés **61**

5 Vers une optimisation de la différenciation : le cas des petites Collections **87**

6 Conclusion générale **101**

Bibliographie **115**

Table des figures **121**

Table des matières **123**

Introduction générale

L'évaluation des apprentissages est une étape fondamentale de la pédagogie. A l'université, dans le but le faire le bilan des acquis des apprenants, des évaluations dites sommatives sont conçues et mises en œuvre par les enseignants à l'issue de chaque cours. Dans la mesure où ces évaluations peuvent amener à l'obtention du diplôme (évaluations dites certificatives), il est primordial que ces dernières permettent un traitement équitable de chacun des apprenants évalués. Dans ce contexte, une contrainte à laquelle font face les enseignants est celle de la limitation de fraude. Pour pallier cela, certains enseignants proposent des sujets différenciés dans une même salle d'examen. L'extension de cette pratique à son maximum consiste à pratiquer une individualisation complète qui consiste à fournir un sujet différent à chaque étudiant lors d'une épreuve d'évaluation.

Toutefois, la différenciation dans les évaluations implique certaines difficultés : l'enseignant qui se trouve parfois dans l'obligation d'avoir une base de questions ou énoncés sources importante en volume pour avoir une différenciation suffisante dans les sujets construits ; au moment de la correction, il peut être plus fastidieux de corriger des copies différenciées que lorsque le sujet est identique pour tout le monde ; la problématique de l'équivalence du niveau de difficulté (équité) entre les sujets car même s'il s'agit d'un même cours pour une même cohorte d'étudiants, le fait que les sujets soient différenciés laisse entendre que l'enseignant doit s'assurer que les étudiants sont tous évalués à un même niveau d'exigence. Dans ce cadre, avoir recours aux environnements informatiques pour automatiser tout ou partie de l'évaluation des apprentissages, est l'une des solutions pour pallier ces difficultés. De nombreux outils numériques proposent en particulier aux enseignants de les assister dans l'étape de conception de leurs sujets. Dans le domaine de recherche des Environnements Informatiques pour l'Apprentissage Humain (EIAH), divers travaux s'intéressent également à l'instrumentation de l'évaluation des apprentissages. Peu d'entre eux en revanche se concentrent sur une conception automatisée de sujets d'évaluation différenciées. Nos travaux de recherche inscrits dans le domaine des EIAH, se focalisent sur cette question, dans le cadre spécifique des évaluations sommatives (plus précisément les évaluations certificatives) en contexte universitaire. En ce sens, le projet DIFAIRT (DIfferentiated FAIR Test) vise à instrumenter l'enseignant dans le

processus complet d'évaluation des apprentissages, en prenant en compte les contraintes de différenciation et d'équité dans des sujets de type Questionnaires à Choix Multiples.

Plus spécifiquement, cette thèse vise à proposer à des enseignants la meilleure approche possible lors de la génération automatique de sujets différenciés à partir d'une base de modèles de questions. Pour cela, elle propose des algorithmes permettant la génération automatique de *Collections* (ensembles) de sujets différenciés de type QCM, avec comme objectif de maximiser la différenciation entre les sujets de la *Collection* générée, tout en minimisant le nombre de questions sources nécessaire à cela. En d'autres mots, être capable de mesurer le niveau de différenciation qu'il peut y avoir entre deux sujets auto-générés, pour favoriser une relaxation du travail de l'enseignant sous deux angles : premièrement, faire qu'il soit capable d'avoir une grande marge de différenciation dans ces sujets sans pour autant être obligé d'avoir une base source avec un volume important de questions ; deuxièmement, proposer des outils pour le faire de façon optimale reposant notamment sur une métrique de mesure de différenciation et des algorithmes de génération adaptés aux divers contextes d'évaluations et besoins définis par l'enseignant. De plus, cette thèse pose les bases d'une algorithmique des graphes, de sorte que les étudiants estimés voisins par l'enseignant obtiennent des sujets avec un niveau de différenciation plus élevé. Pour l'expérimentation de ces travaux, des centaines de *Collections* de sujets sont générés, traités et analysés, dans le but de mesurer les performances de chacun des algorithmes de génération en terme de différenciation.

Les principales contributions de ces recherches ont consisté en la définition d'une métrique pour mesurer différenciation dans les sujets d'évaluation, la proposition de trois approches de génération de collections de sujets à partir d'une base source et la proposition de méthodes d'attribution de petites collections de sujets aux étudiants de façon optimisée lors d'une épreuve d'évaluation.

La suite de ce document est développée à travers 6 chapitres :

Le Chapitre 1. caractérise le contexte pédagogique de ces travaux. Après une définition globale de l'évaluation des apprentissages, un panorama d'un ensemble d'outils d'automatisation des évaluations est dressé. Ce chapitre finit en présentant un exemple d'outils d'automatisation qui est lui-même à l'origine de ce travail de recherche.

Le Chapitre 2. positionne ces travaux dans le domaine des EIAH. Ce domaine de recherche est d'abord défini tout en portant l'emphase sur les EIAH pour les enseignants et l'instrumentation de l'évaluation des apprentissages. Le chapitre est conclu par une présentation du projet DIFAIRT dans lequel s'inscrit cette thèse.

Le Chapitre 3. décrit une métrique originale conçue et développée dans le cadre de ces travaux. Cette métrique nous servira dans la suite d'outil pour mesurer la différenciation des *Collections* générées et pour orchestrer et optimiser la différenciation en cours de génération. La dernière section de ce chapitre étudie les performances en termes de différenciation d'une première méthode de génération de référence et présente les premiers résultats.

Le Chapitre 4. se focalise sur les deux principales méthodes de génération de sujets différenciés qui sont élaborées et expérimentées dans cette thèse. La première méthode est basée sur un algorithme de génération par sélection, la seconde trouve son inspiration dans le domaine des problèmes de satisfaction sous contraintes ici symbolisée par l'optimisation de la meilleure différenciation possible dans une *Collection* de sujets. Pour cette méthode, nous utilisons des méta-heuristiques par l'intermédiaire d'un algorithme génétique dédié. Enfin, ce chapitre dresse les fondements d'un prototype à destination de l'enseignant pour la mise en œuvre optimale de cette génération.

Le Chapitre 5. présente une méthode de génération fondée sur de petites *Collections* de sujets avec une exigence de plus grands niveaux de différenciations. Cette dernière mélange nos méthodes de génération (métrique et algorithmes) avec une approche tirée de l'algorithmique des graphes. Dans ses sections, nous démontrons son importance et son fonctionnement à travers des expérimentations. Également, une approche d'attribution de sujets différenciés dans les salles de classe qui est basée sur la coloration de graphes est présentée.

En dernier lieu, une conclusion récapitule ce travail de recherche et présente les perspectives de recherche en introduisant de premiers travaux menés sur la contrainte de l'équité entre les sujets d'évaluation générés, principalement basés sur les analyses statistiques des résultats des évaluations précédentes.

Chapitre 1

Contexte pédagogique

Nos recherches trouvent leur contexte pédagogique dans l'évaluation des apprentissages au niveau de l'enseignement supérieur. L'évaluation peut se présenter sous diverses formes et à différents moments de l'apprentissage. Ainsi, ce chapitre vise en premier lieu à mieux comprendre sa place, son importance et les contraintes qu'elle peut engendrer sur les enseignants et apprenants. Les différents axes selon lesquels l'évaluation des apprentissages peut se voir instrumenté grâce au numérique sont également étudiés.

Ainsi, la première section présente l'évaluation des apprentissages en portant l'emphase sur son rôle dans l'enseignement supérieur. La deuxième met en avant certains outils numériques d'assistance aux évaluations. La dernière section présente le prototype YMCQ, développé au sein de l'équipe de recherche. Ce dernier permet de faire une automatisation quasi complète du processus d'évaluation pour les Questions à Choix Multiple (QCM) en proposant une composition sur papier et une différenciation des sujets.

1.1 L'évaluation des apprentissages

D'après le dictionnaire Larousse, évaluer est défini comme *"l'action de déterminer la valeur de quelque chose"* [38]. Linternaute parle de l'estimation de la valeur, du nombre, de l'importance ou de la grandeur des choses [43]. D'un point de vue plus général, Neil Postman présente l'évaluation comme élément inévitable dans la communication humaine [58]. Dans le cadre spécifique de la pédagogie, c'est l'apprentissage, les acquis ou les compétences des apprenants que l'on vise à mesurer ou estimer. Dans le compte-rendu [29], Hadji Charles présente l'évaluation comme *"la mise en relation des éléments issus d'un observable appelé référé et un référent pour produire de l'information éclairante sur le référé afin de prendre des décisions"*. Le référent représente les compétences que l'apprenant devrait acquérir et le référé étant celles qu'il a réellement acquises. Ainsi, la décision à l'issue d'une évaluation peut être de passer d'une leçon à une autre ; de passer d'une

année scolaire à une autre ; d'obtenir un diplôme universitaire ou non ; etc. Ces différentes définitions accordent aux évaluations une grande importance car elles influent sur la progression d'un parcours, comme valider une compétence ou une matière ; valider un module pour passer au suivant ; ou encore plus généralement en contexte universitaire, refaire une année d'étude ou débuter une carrière professionnelle.

Dans cette section, nous faisons un tour d'horizon sur l'enseignement supérieur et les types d'évaluations qui sont les plus souvent utilisées. Une emphase est portée sur l'évaluation certificative, type d'évaluation auquel ces travaux de recherche s'intéressent plus particulièrement.

1.1.1 Enseignement supérieur

L'enseignement supérieur en France est caractérisé par sa division en deux grands groupes qui sont les universités et les grandes écoles. L'université a connu beaucoup de changements depuis les années 60 et parmi ces changements, on peut parler de la nouvelle organisation du cycle universitaire Licence Maîtrise Doctorat (LMD) [1] ou encore l'arrivée massive des nouvelles institutions. D'après le Ministère de l'Enseignement supérieur, de la Recherche et de l'Innovation [63], l'enseignement supérieur en France respecte l'organisation du LMD [2]. Celle-ci a été adoptée dans le cadre de l'harmonisation des cursus universitaires au niveau de l'Europe, voulant que l'université soit organisée autour de trois principaux diplômes. Ainsi, la licence correspond au trois premières années d'études. Le diplôme de Master implique deux années supplémentaires avec pour objectifs, soit de préparer les étudiants pour un parcours doctoral, soit pour une qualification et insertion professionnelle de haut niveau. Enfin, le doctorat correspond à un bac + 8 ans d'études.

Dans les nombreux pays Européens adoptant le schéma LMD, notamment en France, une année universitaire est divisée en deux semestres. A chaque unité sont affectés 30 crédits ECTS (European Credits Transfer System) [1], donc au terme d'un diplôme de licence un total de 180 crédits est validé. L'étudiant qui va jusqu'au Master cumule au total 300 crédits, équivalant à 10 semestres. Une unité est constituée d'un ensemble d'UE (Unités d'Enseignements) pouvant elles-mêmes être déclinées en EC (Eléments Constitutifs). Les enseignements prodigués le sont dans la grande majorité des cas sous la forme des cours magistraux, complétés ou non de séances de travaux dirigés et/ou de travaux pratiques. Pour chaque UE, et donc par extension pour chaque diplôme, des Modalités de Contrôle des Connaissances (MCC) sont définies. Ces MCC rentrent dans un cadre général fixé par les textes ministériels, puis sont réadaptées par les enseignants selon des critères

1. Un système de pondérations développé par l'Union Européenne ayant pour objectif de faciliter la lecture et la comparaison des programmes d'études universitaire au sein des différents pays européens.

pédagogiques mais également selon des critères organisationnels (taille de la cohorte, les salles envisagées, le nombre d'intervenants, etc). Elles permettent d'établir le nombre de sessions proposées aux étudiants, la nature des évaluations proposées (exemple : oral, écrit), leurs nombres ainsi que les règles de calcul permettant d'établir la validation d'une UE et l'attribution des crédits ECTS correspondant. Un jury est réuni à chaque session pour étudier les résultats obtenus par les étudiants et vérifier les décisions de validation en regard des MCC. Dans ce contexte, le rôle des enseignants est, pour les UE ou EC d'UE dont ils ont la charge, de concevoir les évaluations qui sont passées par les étudiants et en produire les énoncés, d'être garant du bon déroulement de ces dernières, et d'organiser et mettre en oeuvre la correction ou la traduction d'un travail rendu en note permettant la décision de jury. Il est également de son devoir de garantir l'équité, ce qui passe notamment par limiter les possibilités de fraude entre étudiants.

Ainsi, l'évaluation joue un rôle déterminant et la question de bien évaluer les apprenants est récurrente, car le but final de toutes les institutions d'enseignement est de former pour enfin certifier les acquis des apprenants. Dans la section suivante, nous présentons les types d'évaluations les plus présents dans le milieu universitaire.

1.1.2 Typologie des évaluations

Dans le milieu éducatif, il existe plusieurs types d'évaluations et chacune d'elle à son utilité propre. Parmi les plus pratiquées à l'université, on peut citer les suivantes :

- **L'évaluation diagnostique** permet, au démarrage de certaines formations, à l'enseignant (et/ou l'apprenant) de prendre connaissance du niveau de compétence des apprenants pour une meilleure adaptation de la stratégie pédagogique [32]. Elle ne pénalise pas les apprenants qui la passent, mais à son issue, l'enseignant dispose d'un point de comparaison, ce qui lui permet de mieux suivre la progression de chacun par rapport au début de la formation. Par exemple, Eduscol [19] du ministère de l'Éducation Nationale propose "Banqu'outils pour l'évaluation" qui rassemble un ensemble d'outils d'aide à l'évaluation diagnostique pour les enseignants du primaire et secondaire. Elle est assez souvent confondue avec l'évaluation pronostique qui est une évaluation d'orientation de plus en plus utilisée à l'université. Cette dernière se fait généralement en début d'année académique pour aider les nouveaux apprenants à mieux choisir leurs cursus [57]. Elle s'apparente à l'évaluation diagnostique, mais elle a un objectif plus global et n'est pas vraiment incluse dans la stratégie pédagogique d'une formation.

- **L'évaluation formative** se fait généralement tout au long d'une formation. Pour citer Linda Allal [3], l'évaluation formative peut-être proactive, interactive ou encore rétroactive. Ainsi, sur la base d'interrogations, d'échanges à l'oral ou de simples observations, on

peut recueillir assez d'informations nécessaires pour valoriser les efforts des apprenants par des encouragements dans le but qu'ils progressent. Ce type d'évaluation permet de prendre connaissance du niveau d'acquis de l'apprenant et de ses difficultés pour une remédiation dans la majorité des cas ou une assistance individuelle [68]. Cette évaluation contribue à l'apprentissage en donnant droit aux essais et aux erreurs aux apprenants [51], elle leur permet de s'auto-évaluer et de définir leurs propres stratégies d'apprentissage. Par exemple, l'outil dénommé Formative [59] propose à des enseignants de créer des sujets d'évaluation d'entraînement pour leurs étudiants comme des quiz de mathématique. On en retrouve également dans les Massive Open Online Course (MOOC) et dans d'autres dispositifs d'apprentissage en ligne tels que les plateformes pédagogiques ou LMS (Learning Management Systems) disponibles dans les universités.

- **L'évaluation sommative** vise à faire un bilan des savoirs et compétences acquis par les apprenants [21]. Il s'agit du type d'évaluation le plus répandu dans l'enseignement scolaire et universitaire, pouvant intervenir à la fin d'une formation ou sur toute la formation à travers plusieurs épreuves et exercices dont les résultats se compléteront. A côté de l'objectif d'attribuer aux apprenants une note, il s'agit d'une obligation ayant pour but de rendre compte à l'administration de l'institution de la performance globale des apprenants [51]. Les évaluations partielles et finales des unités d'enseignements dans les universités sont des exemples significatifs d'évaluations sommatives.

- **L'évaluation certificative** rentre en jeu quand l'évaluation sommative a pour but final de décerner un certificat, une attestation ou un diplôme aux apprenants. Il s'agit du jugement dernier des apprenants pour une formation suivie [56]. Si l'on prend l'exemple d'élèves de terminale, ils passent toute l'année des évaluations sommatives permettant aux enseignants de vérifier leurs acquis dans les matières enseignées. En revanche, l'évaluation qui est certificative est le Baccalauréat. A l'université, l'ensemble des évaluations sommatives, si leur note entre dans les MCC des UE, peut être considéré comme certificatif puisqu'il délivre les crédits ECTS correspondants, crédits qui sont des parts du diplôme final. C'est pourquoi concevoir et organiser des évaluations à l'université oblige à beaucoup de rigueur puisque ce qui est à la clé c'est l'obtention du diplôme par l'étudiant, mais aussi la valeur de ce diplôme.

La section suivante apporte un focus sur l'évaluation des apprentissages à l'université.

1.1.3 Évaluer en contexte universitaire

Dans le milieu universitaire, l'évaluation des apprentissages joue un rôle clé. Le ministère de l'Enseignement Supérieur met l'accent sur l'importance de bien maîtriser les

évaluations à l'université car l'impact se verra sur la qualité des études. Pour citer "L'évaluation des étudiants à l'université : point aveugle ou point d'appui ?" [65], *"l'évaluation des étudiants à l'université devrait être considérée comme stratégie pour la qualité des études, les examens universitaires devraient clarifier leur modèle d'évaluation car on est loin de toujours savoir quelles compétences attestent un diplôme"*. C'est l'enseignant qui décide quels types d'évaluations utiliser dans le cadre de son cours. Certains peuvent mettre en place seulement des évaluations sommatives, comme d'autres peuvent mettre en place pour un même cours à la fois des évaluations formatives, sommatives, voire une évaluation diagnostique. Et pour différentes raisons comme la taille de cohorte d'étudiants en salle de classe, ou encore le format d'un cours, chaque enseignant s'appuie sur la méthode qui convient le mieux pour l'évaluation des apprenants.

Il est de plus en plus fréquent que les universités imposent aux nouveaux étudiants une évaluation pronostique. Une interprétation possible est qu'étant fraîchement diplômés du baccalauréat, ils ne sont pas forcément conscients de leurs aptitudes à suivre un parcours dans une filière donnée. Une deuxième interprétation est le fait que d'années en années les cohortes d'étudiants qui arrivent à l'université ne cessent d'augmenter. Cette évaluation se fait dans certaines universités sous supports numériques, dans d'autres sous supports papiers dans des salles d'examen. Une fois que l'apprenant intègre sa filière, les autres types d'évaluations rentrent en jeu.

Pour ce qui concerne l'évaluation sommative à l'université, les processus peuvent varier d'une institution à une autre et même d'une UE à une autre. En premier lieu on retrouve le modèle comportant deux évaluations sommatives, appelées examen partiel et examen final. Il existe aussi le contrôle continu pour certaines unités d'enseignement. Cela suppose qu'une suite d'évaluations se tient tout au long du déroulement de la formation. Ces évaluations peuvent-être individuelles ou collectives et peuvent se tenir durant les différentes séances de travaux pratiques et travaux dirigés. Pour autant, la combinaison des notes obtenues à ces évaluations caractérise le niveau de l'apprenant. Quoiqu'il en soit, qu'il s 'agisse d'examens ou de contrôle continu, la méthode d'évaluation adoptée est déterminée par l'enseignant. Il peut s'agir d'un devoir surveillé écrit ou d'un oral dans le cas d'examens, et plus largement d'épreuves de type TD ou TP notés, devoirs de maison, rapports ou mémoires à rendre, projets. Les devoirs surveillés peuvent prendre diverses formes telles que l'étude de cas, questions ouvertes, dissertations, exercices ou encore Questions à Choix Multiples. Dans tous les cas, les corrections des épreuves et l'attribution des notes incombent aux enseignants qui les ont conçues. De plus, ces épreuves peuvent se faire sur papier ou encore sur support numérique, ce qui rend l'organisation plus complexe lorsqu'il s'agit d'épreuves surveillées.

Il est important de préciser que les QCM sont assez présents dans les évaluations à l'université. Ce type de sujets offre l'avantage d'être relativement facile/rapide à mettre en place pour les enseignants. D'autre part, les QCM permettent de garder une objectivité surtout au moment de la correction des copies. L'enseignant qui fait sa formation pour une grande cohorte d'étudiants met en place généralement des sujets de type QCM qui offrent l'avantage d'une correction moins fastidieuse que s'il s'agissait des questions ouvertes. Pour nos travaux, nous avons fait le choix de commencer par ce type de sujets pour son avantage d'être simple à automatiser, comparativement aux autres types de sujets. Mais aussi, ce un des type d'évaluation qui est assez utilisé pour faire de la différenciation en salle d'examen.

1.1.4 La différenciation dans les évaluations

Les enseignants sont amenés à concevoir régulièrement des épreuves d'évaluation. Dans ce contexte, la création de sujets d'évaluation différenciés (dans le contenu et le placement des éléments) consiste notamment à s'appuyer sur un ensemble de sujets, questions ou exercices capitalisés, qu'ils réutilisent et réadaptent d'une épreuve à l'autre. Dépendamment de la nature du cours en évaluation, différents arguments peuvent être avancées par les enseignants [13]. Parmi ceux qui reviennent le plus souvent :

Une seule cohorte d'étudiants évalués en simultané : en contexte universitaire, la majorité des épreuves d'examen de type sommatif se font en salles de classe physiques. Selon la topologie de la salle, du volume de la cohorte d'étudiants, de la nature du cours, etc. les apprenants sont assis plus ou moins proches les uns des autres. Se basant sur le fait que l'évaluation permet de mesurer des compétences acquises dans l'objectif de prendre une décision (à partir de la note de passage), il est important que tous les apprenants soient évalués à un même niveau de rigueur et d'exigence. Pour assurer cette équité entre étudiants, on doit être sûr que les apprenants ont tous été évalués de façon individuelle, donc sans possibilité de tricheries ou fraudes. En effet, d'après le rapport "Rapport - n° 2007-072" du Ministère de l'Enseignement Supérieur, de la Recherche et de l'Innovation, la fraude reste massive à l'université. Environ 50% des étudiants la pratique. Ce même rapport a permis de relever quelques causes telles que des surveillants qui ferment les yeux ; des convocations en conseil de discipline qui sont trop rares ; plagiat accru par l'accès à Internet ; etc. Pour limiter la triche l'enseignant a des options telles que, faire des sujets difficilement fraudables (ex : la dissertation). Le problème est que ce genre de sujets implique des corrections très longues. L'enseignant peut aussi exiger une surveillance adéquate à la topologie de la salle d'examen, ce qui implique une grande mobilisation de ressources humaines lors de l'épreuve. La possibilité qui parait être la plus faisable est la différenciation des sujets, c'est-à-dire proposer plusieurs sujets "équivalents" distribués alternativement. Lorsqu'il s'agit d'un ensemble de questions à choix multiple,

la différenciation peut se faire en mélangeant les questions et les réponses.

Une même UE/EC d'UE pour plusieurs cohortes d'étudiants sur une même session : à cause du volume important d'une promotion, les étudiants peuvent se voir subdivisés en des sous-cohortes pour suivre un même enseignement dans des créneaux distincts. C'est le cas des groupes de travaux dirigés et de travaux pratiques dans les universités. Ce mode opératoire peut parfois se voir répété au moment des évaluations, où chaque sous-cohorte d'étudiants passe son épreuve d'évaluation dans des créneaux horaires différents. Ceci peut être dû à de simples problèmes logistiques comme l'indisponibilité de salle d'examen pouvant contenir toute la promotion. Ceci peut d'autre part être dû à un choix qui rentre dans la stratégie pédagogique. Si l'enseignant prépare un seul et même sujet d'évaluation pour les étudiants, ceux de la sous-cohorte du premier créneau peuvent facilement communiquer le contenu du sujet d'évaluation à ceux du deuxième créneau et ainsi de suite. Dans ce cas, on se retrouve encore dans une configuration où tous les étudiants d'une même promotion, ne seront pas évalués avec un même niveau d'exigence et de rigueur. Pour pallier à cela, un des meilleurs moyens reste la différenciation de sujets d'évaluation. Pour ça, l'enseignant prépare un sujet différencié pour chaque sous-cohorte, et il peut aller jusqu'à préparer un sujet différent pour chaque étudiant de la promotion.

Par ailleurs, d'autres raisons telles que les infrastructures disponibles, (exemple : salles, tailles des cohortes) ou encore les stratégies pédagogiques choisies, peuvent pousser l'enseignant à travailler sur la conception de sujets différenciés. L'axe principal des travaux de recherche de cette thèse est d'étudier les possibilités pour assister à l'aide d'outils numériques l'enseignant dans la conception ces sujets différenciés. Après avoir dressé un bilan de l'évaluation des apprentissages en contexte universitaire, et établi un focus sur la différenciation des sujets d'évaluation, la section suivante présente un panorama d'outils numériques au service de l'évaluation des apprentissages.

1.2 Outils au service de l'évaluation des apprentissages

Dans les universités, la conception des sujets d'évaluation peut devenir extrêmement complexe, fréquente voire répétitive. Dans ce contexte, de nombreux outils numériques permettent d'instrumenter certaines étapes de l'évaluation des apprentissages. Cette section permet de catégoriser certains outils d'évaluation qui servent aux enseignants et d'autres qui sont dédiés aux apprenants. La fin de la section traite de la question de la normalisation.

1.2.1 Outils d'évaluation à destination des apprenants

Divers outils numériques servent de support à l'activité d'évaluation des apprentissages, coté apprenant. Certains outils comme les LMS (Learning Management System) offrent des possibilités de passer des épreuves en ligne ou de soumettre des sujets d'évaluation de types projets, dissertations, questions ouvertes, etc. Ces systèmes proposent des espaces numériques de travail tout en implémentant des modules de gestion de différents types d'épreuves pour des UE diverses. Ils respectent généralement certaines normes d'e-learning comme LOM (Learning Object Metadata), ce qui confère une interopérabilité entre eux. Comme exemple de LMS, on peut parler de Moodle qui est mondialement reconnu et qui inclut des fonctionnalités permettant la mise en place des épreuves de différents types auxquelles les étudiants accèdent directement depuis leurs propres comptes [49]. Il est très souvent utilisé dans des universités, où les étudiants se voient les utiliser pour suivre des enseignements mais aussi pour composer [27]. Dans un tel LMS, l'enseignant a la possibilité de mettre en place une évaluation de type QCM. Pour ce faire, 3 grandes étapes à suivre : préparation de son projet de QCM et importation des questions dans son espace ; paramétrage et création de l'activité « Test » ; et association à l'activité « Test » des questions de son espace. Une fois que tout est en place, il ne lui reste qu'à activer le « Test » pour que ses apprenants y aient accès. Du point de vue de la différenciation, Moodle offre une fonctionnalité de mélange des questions et des réponses de manière aléatoire [50]. Donc, les étudiants auront les mêmes contenus, dans un ordre différent. ¨Pour ce genre de Test, les étudiants peuvent avoir un accès instantané aux résultats après composition, contrairement aux autres types de questions plus ouvertes qui nécessitent une assistance humaine pour la correction. Moodle implémente également l'évaluation par les pairs.

Outre des LMS, nombreux étudiants suivent des formations sur des dispositifs d'apprentissage en ligne et généralement ils sont forcés d'utiliser les systèmes d'évaluation automatisés liés à la plateforme en question. Par exemple, des plateformes d'apprentissages en ligne [53] [14] utilisent quasiment les mêmes types d'évaluation sus-cités pour leurs étudiants, à savoir une évaluation d'orientation au début, des évaluations formatives pendant et des évaluations sommatives/certificatives à la fin de la formation. Éventuellement, on y retrouve également l'évaluation par les pairs ou encore l'évaluation semi-automatisée dans lesquelles, seulement les évaluations intermédiaires se font en ligne. Comme exemple de plateforme, on peut citer OpenClassRooms qui très reconnue en France et qui propose un ensemble de formations certifiantes aux étudiants [66]. En terme de différenciation, on retrouve quasiment toujours la même méthode de mélange des questions et réponses de manière aléatoire.

1.2.2 Outils d'aide aux évaluations pour les enseignants

Coté enseignant, les outils servant à la conception et à la mise en ligne d'évaluations sont les mêmes que ceux destinés aux apprenants. A la différence près que coté enseignants, les LMS permettent d'apporter une assistance dans la création des épreuves tout en facilitant la réutilisation d'épreuves existantes. Par ailleurs, les LMS offrent la possibilité aux enseignants d'observer l'activité des apprenants, via les Learning Analytics ou de collecter les travaux effectués par les étudiants. Les plateformes facilitent également l'accès aux statistiques de réussite des étudiants lors des épreuves à correction automatisée. Parmi ces outils, une grande quantité s'intéressent aux épreuves de types Questionnaires à Choix Multiples [39].

Concernant la conception des QCM en particulier, un grand nombre d'outils auxquels nous nous sommes intéressés sont payants. EvalBox [22] propose aux enseignants de créer des sujets d'évaluation en ligne pour une composition en ligne ou sur papier. Cet outil présente l'avantage du service de la correction automatique et de l'archivage en ligne, ce qui allège l'enseignant de certaines tâches. Il s'agit d'un des outils les plus utilisé par les enseignants. En 2015, 100 000 sujets ont été générés. Nous pouvons également citer l'outil Web-MCQ qui est développé à l'université de Bolton [31]. Il est très proche de EvalBox en terme de fonctionnalités. Il propose aux enseignants de créer en ligne leurs sujets d'évaluation. Ainsi, les étudiants peuvent se connecter à leur tour sur la plateforme pour composer et l'enseignant reçoit les résultats en temps réel [60]. Le fait d'être multiplateformes et adaptable aux différents médias contribue à son utilisation massive. Cependant, il nécessite un abonnement mensuel.

Par ailleurs, on peut citer l'outil Formative 1.1.2 qui permet de poser des questions formatives aux apprenants en direct (pendant le déroulement d'un cours). Sa version gratuite permet d'accéder à des types de questions comme les QCM, les Vrai ou Faux, etc. L'outil Beekast [7] se positionne comme une plateforme interactive intégrant un espace d'échange dans le but d'encourager la prise de parole, d'évaluer et faciliter la prise de décision. Il offre pratiquement les mêmes fonctionnalités que Formative aux enseignants et est également adapté aux grandes conférences.

L'outil QCM-Direct offre une solution d'automatisation des évaluations en se basant sur une composition papier. Il permet d'importer des contenus à savoir des questions et réponses pour la génération automatique des sujets. L'outil se charge du calibrage et de l'impression des réponses [16]. La force de QCM Direct réside dans sa solution de correction automatique de sujets QCM sur papier, et ses retours d'analyse statistique des résultats. A l'UPJV, il est surtout utilisé par l'UFR de Médecine pour le concours de la Première Année Commune des Études de Santé (PACES), ou par l'UFR des Sciences et

Techniques, des Activités Physiques et Sportives (STAPS). Quant au mode de fonctionnement de QCM Direct c'est l'enseignant qui s'occupe en amont de créer le sujet (fichier .doc ou .txt qui contient les questions et réponses) en respectant certaines règles. Il prépare également la liste des candidats, le barème à respecter et le mode d'identification des candidats (ex : numéro de la carte d'étudiant, code barre, etc.).

A part les LMS comme Moodle, les autres outils d'aides sus-cités ont des points communs en termes de restriction de l'autonomie de l'enseignant utilisateur. Par exemple, l'enseignant n'a pas forcément le total contrôle de ses données (comme les questions sources ou encore les résultats d'analyse). Ou encore, la notion de différenciation des sujets d'évaluation n'est pas perçue de la même façon d'un outil à un autre. Certains outils comme QCM-Direct et Eval-Box proposent comme différenciation un mélange aléatoire des contenus dans les sujets générés. Néanmoins, ils ne permettent pas une différenciation totale sur un ensemble de sujets.

Entre les outils d'aide à l'évaluation, on peut trouver pas mal de différences tant au niveau pédagogique qu'au niveau de la structure des données des utilisateurs. Par conséquent, l'enseignant qui veut passer d'un outil à un autre peut se trouver face à la situation de devoir reconstruire ou reformater son ancienne banque de contenus sources pour s'adapter au nouvel outil. Ceci est un problème d'interopérabilité entre les outils existants. Pour cela, des normes ont vu le jour avec l'objectif d'uniformiser le format des contenus d'évaluation (questions et réponses) pour faciliter les échanges de contenus entre les différents systèmes de gestion des évaluations. La sous-section suivante présente les normes les plus influentes dans l'automatisation des évaluations.

1.2.3 Outils d'évaluation et interopérabilité

Les systèmes d'évaluation sont soumis à certaines normes conventionnellement acceptées depuis quelques années. Une des organisations reconnues dans le développement des normes d'apprentissage et de technologie éducative est IMS-GLC (Instructional Management System – Global Learning Consortium) [25]. Cette organisation existe depuis les années 2000, elle est collaborative à but non-lucratif et réunit environ 150 organismes. Elle a réalisé de multiples efforts dans la normalisation et le développement d'un écosystème en faveur de l'interopérabilité entre les systèmes éducatifs. Dans cet écosystème, nous pouvons citer des initiatives comme :

- LEARNING DATA AND ANALYTICS : qui permet le partage de données via différents types de plateformes digitales en temps réel entre des acteurs comme les étudiants, les enseignants et chercheurs.
- LEARNING PLATFORMS, APPS AND TOOLS : qui travaille sur l'inclusion des contenus digitaux, des outils d'aides à l'apprentissage et des supports d'évaluation

dans les institutions.

- INTEGRATED ASSESSMENT : qui propose des normes pour permettre l'accessibilité, la pérennité et l'intégrité des contenus des différents types d'évaluation. Dans cette initiative qui s'intéresse aux évaluations, des normes comme Computer Adaptative Testing (CAT), Accessible Portable Item Protocol (APIP), ou encore Question and Test Interoperability (QTI) sont proposées, toutes dans l'objectif de structurer et de caractériser les ressources digitales dédiées aux évaluations.

Ainsi, la norme IMS-QTI (Question and Tests Interoperability) permet spécifiquement l'interopérabilité entre les éléments constituant des différents systèmes de gestion des évaluations [61]. Cette norme garantit des avantages comme la facilitation de l'accès aux ressources pour les enseignants et étudiants, la flexibilité entre les évaluations sur papier et en ligne, la facilitation d'échange de données entre différents systèmes d'évaluation, la manipulation facile des résultats et leur analyse, etc. L'objectif premier de QTI est de faciliter l'échange des éléments constitutifs d'une évaluation entre différents environnements d'apprentissage.

Dans la norme IMS - QTI, un sujet d'évaluation (un Test) est un composant qui est fait d'un ensemble de blocs (Testparts) comme dans la figure 1.1. Chaque Testpart contient un ensemble de Sections. Chaque Section est faite d'un ensemble d'Items. Ainsi, la plus petite granularité dans un test d'après cette norme est l'item [15]. L'Item lui-même est un bloc contenant un énoncé à destination de l'apprenant et les réponses possibles ou proposées pour cet énoncé.

Un prototype permettant l'automatisation du processus d'évaluation par QCM sur la base d'une composition sur papier a été élaboré au sein du laboratoire MIS. Ce dernier, nommé YMCQ (whY Multiple Choice Question), constitue la genèse de ces travaux de recherche et est présenté dans la section suivante.

1.3 YMCQ, un exemple d'automatisation

YMCQ permet de générer automatiquement des ensembles (collections) de sujets d'évaluations tous différenciés pour des sessions d'évaluation de type QCM en permettant notamment une gestion des processus d'évaluation avec passation sur papier [35]. Il permet également de corriger automatiquement les copies des étudiants, et de faire une notation automatique en suivant le barème prédéfini. Le cycle de fonctionnement de YMCQ compte 3 étapes :

— La conception d'une base de questions sources ;
— Le paramétrage et la génération automatique des sujets d'évaluation ;

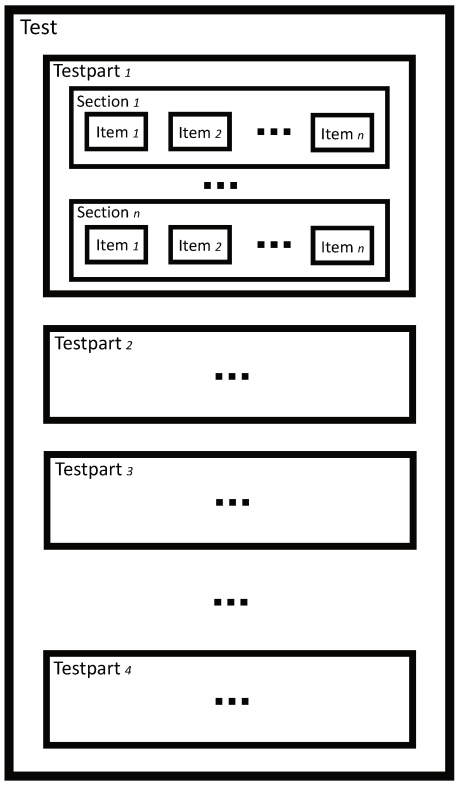

Figure 1.1 – Schéma de QTI représentant la décomposition d'une évaluation

Q : Qu'est-ce que le discours de la méthode ?

- Un célèbre discours prononcé par Galilée au sein de l'assemblée nationale le 12 janvier 1655

+ Un écrit de René Descartes

+ Un ouvrage fondateur de la démarche scientifique

- Un écrit de Blaise Pascal

- Un écrit d'Isaac Newton

FIGURE 1.2 – Format des questions sources

— La correction automatique des copies des étudiants après leur numérisation ;

Chacune de ces étapes est détaillée dans les sections suivantes.

1.3.1 Conception d'une base source

Pour générer des collections de sujets d'évaluation différenciés, YMCQ s'appuie sur un ensemble de modèles à partir duquel il construit des séries de questions à choix multiples. Ces modèles sont regroupés dans une base de questions sources. Chaque question source est composée d'un énoncé, puis de toutes les réponses possibles (bonnes ou mauvaises) envisagées par l'enseignant pour cet énoncé, sachant que le nombre de réponses possibles associées à un énoncé de question n'est pas borné. Cette tâche est réalisée par l'enseignant dans un fichier suivant un format descriptif élémentaire. L'ajout de modèles de questions se fait suivant un format bien défini. Ainsi, la figure 1.2 illustre par l'exemple le format utilisé pour décrire les trois éléments (q, bonnes réponses, mauvaises réponses) constituant chaque modèle de question. La première ligne comporte un énoncé et débute par la séquence "`Q:`". Elle est exprimée sous la forme d'un texte brut. A chaque ligne non vide suivant l'énoncé de la question correspond un élément de choix possible de mauvaises et de bonnes réponses. L'appartenance à l'un ou l'autre des ensembles se fait par le caractère débutant la ligne. Ainsi, les caractères "+" et "-" indiquent que le choix est une réponse valide ou invalide et appartient par conséquent à l'ensemble des bonnes ou mauvaises réponses. Par la suite, on notera 'N' le nombre de questions sources appartenant à une base.

Il est à noter qu'à cette phase de constitution de la base source s'associe naturellement une capitalisation pour une réutilisation lors de différentes sessions d'examens.

1.3.2 Génération de sujets différenciés

La génération automatique d'un ensemble de sujets QCM débute par une phase de paramétrage permettant de dimensionner l'épreuve d'évaluation. Ce dimensionnement consiste à positionner trois paramètres associés à l'évaluation d'un groupe d'apprenants à savoir : le nombre de sujets QCM à générer (noté m), le nombre de questions à choix multiples (noté p et $p \leq n$) par sujet et le nombre de choix possibles par question dans l'ensemble des sujets (noté x). Ici, nous supposons que tous les sujets sont de même longueur et que chaque question contient une même quantité de choix de réponses.

La génération automatique suit, pour chacune des copies spécifiées par l'enseignant, le processus suivant :

— A partir de la base de modèles de questions, YMCQ procède à un tirage aléatoire sans remise de p énoncés de questions. L'ordre du tirage détermine l'ordre des questions. Pour chaque question q_i ($1 \leq i \leq p$) issue de ce tirage, la sélection de l'ensemble des choix multiples suit un processus quasi similaire.

— Un premier tirage aléatoire et sans remise du choix noté x_{ref} est réalisé sur l'ensemble des choix de réponses correctes. Ce premier tirage garanti qu'au moins une réponse correcte est associée à q_i.

— Au terme de cette première étape on procède à un tirage aléatoire sans remise de $x-1$ choix dans l'ensemble des choix restant afin de compléter l'ensemble de choix multiples associé à q_i.

— A l'issue de cette sélection de choix multiples parmi lesquels au moins l'un d'entre est valide, YMCQ procède à un mélange aléatoire suivant l'algorithme de Fisher-Yates [8].

Vient ensuite une phase de génération des PDF avec des codes-barres d'identité pour chaque sujet et sur chaque page de chaque sujet. Ainsi, on obtient trois groupes de fichiers par génération qui sont les sujets, leurs corrigés sous un format PDF et un fichier d'historique de la génération. La section suivante présente la méthode de correction automatique de YMCQ.

1.3.3 Correction automatique

Après une numérisation de toutes les pages des copies (peu importe l'ordre), la correction consiste à identifier chaque page de chaque copie grâce à son qr-code associé. YMCQ procède ensuite à une détection de marques. Chaque marque détectée est comparée au masque liée à la page lors de la phase de génération. Comme il est représenté dans la figure 1.3, la page à gauche est la copie de l'étudiant qui est comparée avec celle de droite, le corrigé. Trois cas peuvent se produire :

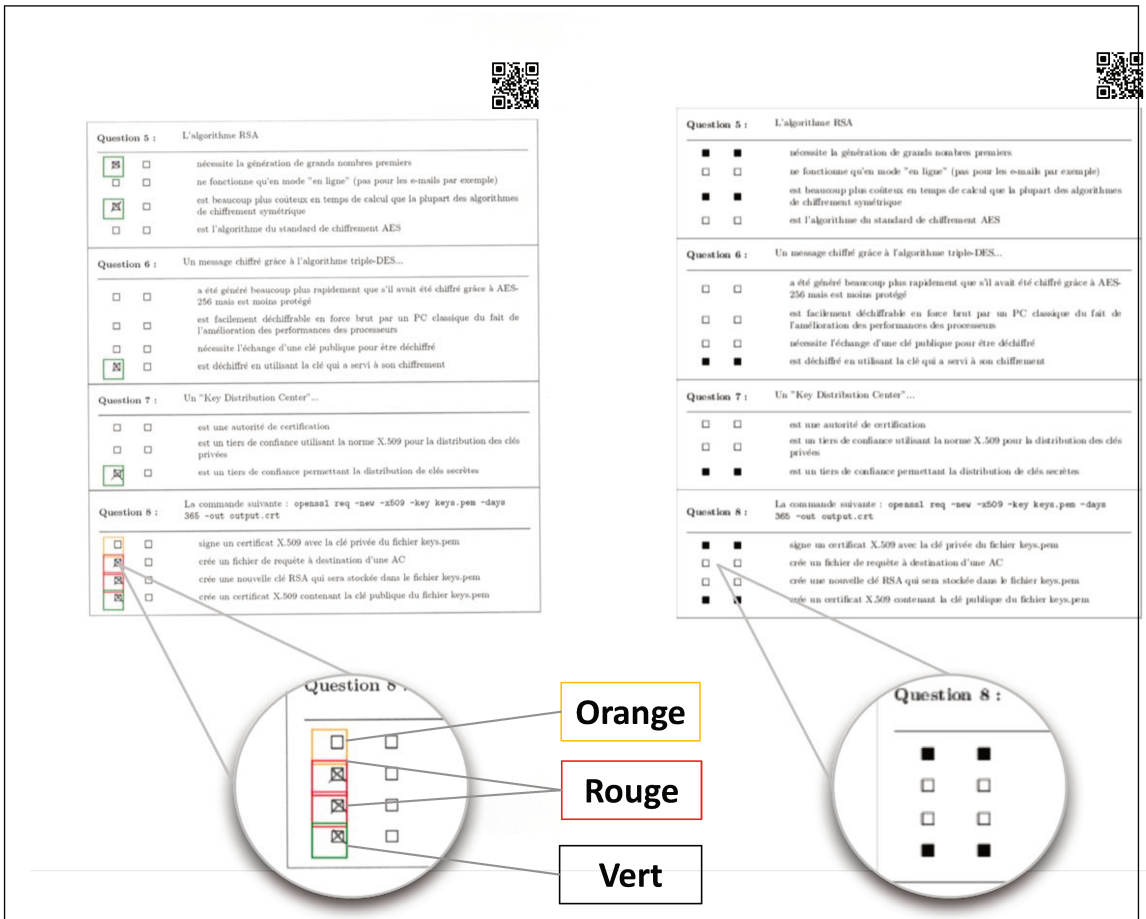

Figure 1.3 – Exemple de correction avec YMCQ

- La marque détectée a été définie : la réponse est correcte et la zone est indiquée en vert
- La marque détectée n'a pas été définie : la réponse est fausse et la zone est délimitée en rouge.
- La marque n'est pas détectée alors qu'elle devrait l'être : la réponse est oubliée et la zone est délimitée en orange.

Afin de permettre une phase de vérification «humaine» rapide, l'évaluation complète examinée et notée est regroupée dans un document global dans lequel chaque page orientée paysage est composée d'une page révisée et du corrigé correspondant. Dans ces conditions, un rapide coup d'œil suffit à identifier les problèmes lors de la détection de la marque.

YMCQ est utilisé depuis quelques années par certains enseignants de l'Université de Picardie Jules Verne. En termes de limites, la génération avec YMCQ se fait en lignes de commande et nécessite un fichier source rigoureusement formaté en UTF-8. Également, il ne peut pas contrôler le niveau de difficulté des sujets différenciés en vue de garantir de l'équité entre elles. Aussi, YMCQ ne procure pas pour le moment une analyse statistique/sémantique des résultats après la correction. La dernière limite est qu'il n'a pas de contrôle sur la différenciation qu'il procure, c'est-à-dire qu'il n'est pas en mesure de spécifier à quel niveau deux sujets qu'il génère sont différents. Ce problème de gestion de la différenciation constitue le point de départ de ces travaux de thèse.

1.4 Conclusion

Ce chapitre a permis de placer ce travail de recherche dans son contexte pédagogique qui est l'évaluation des apprentissages. Plus spécifiquement, ce travail pose la question de la différenciation dans les sujets dédiés aux épreuves d'évaluation dans les universités, surtout quand il s'agit d'évaluation sommative/certificative. Ces évaluations sont généralement basées sur des processus complexes qui reposent principalement sur l'enseignant qui les conçoit et les corrige. Aussi, cela leur demande d'être rigoureux et équitables tout en essayant de limiter la fraude. Ainsi, ces derniers se tournent vers des outils numériques permettant d'automatiser tout ou partie du processus d'évaluation.

De nombreux outils d'assistance aux évaluations existent mais ils ne répondent pas tous à la question de la différenciation et ceux qui le font restent à un niveau basique qui est le simple mélange des contenus. YMCQ permet la génération de sujets différenciés pour les épreuves à l'université. Il englobe également les phases de conception et de correction des sujets qu'il génère. Toutefois, il a certaines limites comme : le fait de ne pas pouvoir

contrôler la différenciation dans les sujets générés ; ou encore l'absence d'analyses sur les résultats des épreuves.

Pour repousser ces limites, le projet DIFAIRT ((DIfferentiated FAIR Test)) a pris naissance autour de YMCQ avec le souhait d'aller plus loin dans la garantie de la différenciation, en proposant son optimisation dans les collections de sujets générés dans un même ensemble. Par ailleurs, la question de l'équité étant centrale dans les épreuves certificatives universitaires, ce projet souhaite également que la génération automatique tienne compte de la difficulté des sujets générés.

Pour sa part, cette thèse s'inscrit dans ce projet et a pour objectif de répondre à la question de la génération de collections de sujets d'évaluation différenciés, voire individuels, en permettant de maximiser cette différenciation, sans avoir à augmenter la taille de la base des questions source de l'enseignant.

Avant de présenter plus en détails les tenants et aboutissants du projet DIFAIRT, et la problématique inhérente aux travaux de cette thèse, le prochain chapitre présente le contexte scientifique de ces recherches.

Chapitre 2

Contexte scientifique et objectifs

Comme cela a été évoqué dans le chapitre précédent, nos travaux de thèse s'intéressent à l'instrumentation informatique de l'évaluation des apprentissages. Plus spécifiquement, elle vise à assister l'enseignant lors de la conception/création de sujets d'évaluation différenciés. Le contexte scientifique dans lequel s'inscrivent ces travaux est celui des Environnements Informatiques pour l'Apprentissage Humain (EIAH). Ce domaine a pour objectif d'étudier les situations pédagogiques informatisées et les logiciels qui permettent de les gérer et de les étudier [74]. Ces outils peuvent être d'utilités diverses, mais tous ont pour but de faciliter l'apprentissage humain.

Dans ce chapitre nous présentons les domaines scientifiques de cette thèse et puis nous décrivons ses enjeux et problématiques.

2.1 Les Environnements Informatiques pour l'Apprentissage Humain

Les EIAH sont des environnements informatiques destinés à favoriser les apprentissages chez leurs utilisateurs humains. Ils ont permis depuis des années à produire des prototypes de logiciels intelligents pour l'éducation [67]. Il existe une différence entre la recherche en EIAH et l'ingénierie des EIAH qui sont généralement confondues. L'ingénierie des EIAH se rapporte à une pléiade d'outils développés dans le but d'être utilisé comme support à l'enseignement. Ces outils sont utilisés par des enseignants comme dans le cadre de la préparation des cours ou des examens ; par les étudiants comme des outils pour s'entraîner à faire des exercices ; par des parents comme des outils permettant de suivre la progression de leurs enfants ; etc... La recherche en EIAH quant à elle traite de la création de connaissances autour des situations pédagogiques et de l'utilisation d'outils informatiques pour l'apprentissage humain [75]. Autrement dit, le champ scientifique des

EIAH permet d'étudier des situations pédagogiques informatisées.

Généralement, les environnements informatiques qui sont soumis à un apprenant ou un enseignant pour la réalisation d'une tache sont d'abord étudiés, conçus, développés, donc préparés pour l'apprentissage. Pour cela, ces outils sont souvent des résultats d'une collaboration entre l'enseignant comme prescripteur, l'apprenant comme utilisateur et les chercheurs [33] [73]. Dans sa pluridisciplinarité, le champs de recherche des EIAH permet d'intégrer des travaux issus de divers autres champs de recherche connexes comme toutes les disciplines qui contribuent aux réflexions sur l'apprentissage et la formation. Parmi ces disciplines, on peut retrouver les sciences de l'éducation [75], la psychologie, les sciences de l'information et de la communication, ou encore les sciences du langage (l'ergonomie, la sociologie, etc). Dans le domaine de l'informatique, on peut aussi bien trouver des sous-domaines tels que l'ingénierie logicielle, l'intelligence artificielle au sens large [30], l'ingénierie des connaissances [47].

Les sciences et techniques éducatives existent depuis des années, la sous-section suivante présente une brève historique de son évolution.

2.1.1 Historique des EIAH

Ce domaine de recherche a énormément évolué ces dernières années et plusieurs facteurs peuvent l'expliquer. Un premier facteur peut résider dans le fait que les nouvelles technologies soient de plus en plus accessibles, ce qui fait changer les pratiques des enseignants et des apprenants. Un deuxième facteur peut résider dans le fait que la quantité d'étudiants arrivant chaque année à l'université a grandi si vite que l'automatisation de certaines pratiques dans l'enseignement deviennent quasi inévitable.

Les technologies sont entrées dans le système éducatif depuis très longtemps. L'informatique est arrivée dans l'enseignement avec la création des premiers ordinateurs dans les années 50. Cet enseignement programmé repose sur le principe du découpage des connaissances en unité élémentaires ; de la participation active de l'apprenant ; de la progression gradué de l'apprenant ; de l'accès immédiat aux résultats [10]. Toutefois, une des plus grandes lacunes de l'enseignement programmé a été la limite en terme d'individualisation, ce qui le rendait inadaptable à certains domaines [77]. Environ dix ans après, l'évolution de l'enseignement programmé a permis de naître les premiers didacticiels. Le domaine des EAO (Enseignement Assisté par Ordinateur) a ainsi permis aux ordinateurs et aux didacticiels d'être de plus en plus présents dans le système éducatif [46].

Dans les années 60 ont émergé les recherches sur les Environnements Intelligents Assistés par Ordinateurs (EIAO) [6]. Les EIAO ont permis d'améliorer les EAO en termes

d'interactivité et d'adaptation à l'apprenant, grâce à l'Intelligence Artificielle [42]. Les STI (Systèmes Tutoriels Intelligents) et les Micromondes apparus à leur tour [10] connaissaient une quasi même remarque étant celle d'isoler de plus en plus les apprenants en les refermant dans un monde, où ils pouvaient de moins en moins avoir le support de l'enseignant ou d'autres apprenants. Toutes ces réflexions sur les points faibles des Micromondes et des Systèmes de Tuteurs Intelligent ont conduits à la deuxième version des EiAO avec un changement de définition du sigle en Environnements interactifs d'Apprentissage avec Ordinateur. Cette nouvelle terminologie a voulu qu'on insiste sur le coté Apprentissage, où l'apprenant peut construire lui-même ses connaissances en interagissant avec l'outil. Les recherches étaient plus orientées vers les apprenants ; les enseignants étaient plus présents comme acteurs clé dans la conception des outils.

Enfin, dans les années 90, la dénomination du domaine des Environnements Informatiques pour l'Apprentissage Humain (EIAH) est apparue. Une évolution du sigle qui fait ressortir une ambition plus globale d'améliorer l'éducation grâce aux environnements informatiques. De plus, cela confirme une réelle prise en compte de la pluridisciplinarité de recherche (inclusion des sciences de l'éducation, de la sociologie, etc). L'arrivée des EIAH a permis de garder le meilleur des travaux et concepts passés en améliorant l'interaction entre le pôle de l'informatique et celui de l'enseignement. Dans ce domaine, les thématiques de recherche sont très variées. Si on fait une rétrospective sur les travaux des dix dernières années, on retrouve des thématiques assez répétitives dans les principales communications des EIAH et d'autres plus récentes, ce qui pourrait expliquer l'évolution de ce domaine de recherche. Comme exemple de thématiques, on peut parler des « Jeux sérieux », très présente dans les EIAH, qui généralement est basée sur une simulation du domaine à enseigner avec laquelle l'apprenant-joueur peut interagir ludiquement au moyen d'une métaphore [44]. La thématique des « Traces d'EIAH » est assez présente également. Les travaux de cette dernière sont très variés, mais principalement, ils s'occupent de récolter des traces d'activités sur des dispositifs d'apprentissage et de les analyser, pour des fins diverses (comme la personnalisation) [69]. La liste étant longue, on peut même parler de celle des « Tuteurs intelligents ». Pour en voir encore plus, il faudrait revisiter les différents actes des EIAH, une conférence internationale francophone qui est organisée tous les deux ans et qui constitue un moment privilégié de rencontre entre les chercheurs du domaine. Les trois dernières sont celles des années 2013 [12], 2015 [24], 2017 [28], etc.

Dans la section qui suit, nous portons l'emphase sur certains en EIAH qui s'intéressent principalement à la thématique des évaluations.

2.1.2 EIAH et évaluations des apprentissages

Parmi les travaux en EIAH sur les évaluations, ASKER permet aux enseignants de créer des exercices d'auto-évaluation pour leurs apprenants [40]. Il répond à deux objectifs qui sont, premièrement, de fournir aux enseignants un outil leur permettant de créer une quantité importante d'exercices portant sur les notions à acquérir ; deuxièmement, fournir aux apprenants la possibilité de s'entraîner avec ces exercices en les résolvant et d'en obtenir un diagnostic. Les types d'exercices que propose ce travail sont divers. On peut citer les appariements, les groupements, le Questions à Trous, les QCM, etc. En termes de différenciation, l'enseignant crée un modèle d'exercice et l'apprenant peut demander à ASKER de lui générer des exercices variés issus du même modèle.

Pépite [33], projet EIAH présenté par Stéphanie Jean-Daubias de l'université Claude Bernand à Lyon 1, offre une assistance aux enseignants dans l'évaluation des compétences des élèves en algèbre. Il est bâti sur trois principaux modules dont le module PEPITEST qui propose les exercices aux élèves et recueille leurs solutions ; le module PEPIDIAG qui tient compte de la diversité des questions et réponses des apprenants pour proposer des techniques d'analyse adaptée à chaque catégorie d'apprenant ; et le module PEPIPROFIL qui établit les profils des élèves et les présente aux enseignants. PEPITEST permet de générer des exercices sous forme QCM, mais la différenciation n'entre pas dans les problématiques traitées par ce projet.

Le projet CLAIRE de l'université de Lyon a proposé un EIAH pour la génération d'exercices d'auto-évaluation [11]. Ce générateur permet à l'enseignant de créer des modèles d'exercices. Ces modèles sont instanciés pour pouvoir donner naissance à un certain nombre d'exercices différents. Son point fort est le fait de proposer des types d'exercices indépendants du domaine, ce qui permet son utilisation à différent niveau scolaires et dans des disciplines diverses. Ici, la différenciation peut-être perçue comme le fait de changer les valeurs d'un exercice provenant d'un même modèle. Toutefois, ce générateur ne présente pas de méthodes permettant de contrôler la différenciation entre deux exercices provenant d'un même modèle ou non.

On peut également citer les travaux de thèse de Montalan [48] qui a proposé un système générateur de questions d'évaluation dans le domaine de la comptabilité financière. Ce générateur produit des exercices de format QCM et Questions fermées depuis une banque de questions sources dans un but d'auto-évaluation formative ; les travaux de Guillaume Durand [18] sur la scénarisation de l'évaluation des activités pédagogiques, ayant pour objet la définition d'un modèle de résultats produits par les EIAH qui puisse être utilisable dans le contexte des ENT ; l'outil Sygep [54] qui vise la génération d'énoncés de problèmes dans divers domaines. L'objectif principal a été de pouvoir créer de façon automatique

des exercices adaptés au niveau de l'apprenant. Pour cela, Sygep utilise une grandeur qui détermine l'apprenant, à partir de laquelle le système soit en mesure de générer de façon contrôlé le problème. Toutefois, dans ces trois travaux, la notion de différenciation dans les évaluations n'est pas mentionnée.

En somme, peu de travaux du domaine des EIAH sont dédiés à l'évaluation des apprentissages. Parmi ceux-là, assez peu s'intéressent à l'automatisation des processus de l'évaluation à savoir la conception et la génération de sujets. Pour ceux qui s'y intéressent comme les projets CLAIRE, PEPITE ou encore ASKER, ils ne se préoccupent pas forcément de la différenciation dans les sujets d'évaluations et d'une génération automatique à partir d'une base initiale de collections de sujets différenciés. Dans la section suivante, nous présentons un des champs scientifiques secondaires dans lequel ces travaux de recherche trouvent également leur contexte.

2.2 Problèmes de satisfactions de contraintes

La problématique de la différenciation dans la génération des sujets d'évaluation relève également des problèmes de satisfaction de contrainte. Il est des problèmes pouvant être modélisés sous la forme d'un système d'équations nommés contraintes. Toute solution à ce système est également une réponse possible au problème initial. Une solution à ce problème consiste à trouver une solution à l'ensemble des contraintes. On parle alors de problème de satisfaction de contraintes (CSP) [20] [37]. La plupart des solutions de ces problèmes quand elles existent font partie d'un paysage non convexe et leur résolution suppose alors de payer le tribut inhérent de la complexité exponentielle. Ainsi, on fait souvent appel à des heuristiques ou encore d'autres méthodes d'optimisation pour avoir des solutions approximatives. Comme exemple d'application du CSP, le problème des huit reines [71] consistant à placer huit reines d'un jeu d'échecs sans que les reines ne puissent se menacer mutuellement sur un échiquier de 64 cases ou encore l'organisation de tournoi sous contrainte, les tournées de véhicules, etc...

Dans le domaine des EIAH, on peut également se trouver face à certaines contraintes à satisfaire. Par exemple, un outil permettant de faire des évaluations formatives et adaptatives peut faire face à la contrainte des niveaux de difficulté des exercices, car la difficulté de l'exercice suivant est souvent adaptée en raison de la réponse de l'apprenant à l'exercice précédent. Dans un cas pareil, il sera question de faire en sorte que l'exercice suivant soit d'un niveau de difficulté le mieux adapté possible au niveau de l'apprenant. Les travaux en EIAH incluant du CSP sont rares mais pas inexistants. Un exemple de travail de cette catégorie est celui d'un "Système Bio-Inspiré pour l'Adaptation Pédagogique dans un EIAH", une thèse réalisée par Amina [4] qui s'intéresse à la personnalisation de

l'apprentissage. Ces travaux ont proposé une méthode de calcul d'ADN afin de générer la séquence personnalisée des objets pédagogiques, grâce à un algorithme génétique pour la représentation du profil de l'apprenant.

Dans le cadre de cette thèse, la principale contrainte à satisfaire est celle de la différenciation entre les sujets d'évaluation générés lors d'une séquence. Il est question de trouver le meilleur moyen de générer des collection de sujets les plus différenciés possible, sans nécessairement agrandir la base source. Nous en parlons de manière détaillée dans le prochaine section qui présente le projet dans lequel s'inscrit cette thèse.

2.3 Le projet DIFAIRT

DIFAIRT est un projet de recherche qui fait suite au prototype YMCQ. L'objectif du projet DIFAIRT est d'aller beaucoup plus loin dans l'assistance apportée aux enseignants quant à la gestion de leurs évaluations différenciées. En ce sens, DIFAIRT vise à proposer des mécanismes garantissant un niveau d'équité entre les sujets d'évaluations générés en se basant sur un mécanisme d'apprentissage à partir d'évaluations passées par des apprenants. Par ailleurs, DIFAIRT vise également à proposer des méthodes de génération optimales en termes de différenciation, c'est-à-dire répondant au maximum aux exigences de différenciation solicitées par l'enseignant.

Le projet s'intéresse en premier lieu à l'optimisation de la génération différenciée et à l'équité dans les sujets d'évaluation composés d'items de types QCM, l'idée à terme est que ces travaux se généralisent à d'autres types d'items que des questions à choix multiples. Les travaux du projet DIFAIRT s'inscrivent ainsi dans une recherche en EIAH. A noter, les outils proposés dans le cadre de ces travaux peuvent aussi s'inscrire parmi les outils de gestion d'évaluations par QCM comme nous avons pu les voir au chapitre 1.

La suite de la section présente les principaux composants de DIFAIRT et les objectifs spécifiques de la thèse.

2.3.1 Principe de DIFAIRT

Dans le cadre de DIFAIRT, un sujet d'évaluation est appelé un "Test". L'ensemble des éléments qui constituent un Test proviennent d'une base qui contient des modèles de questions (Patterns). Se référant à la figure 2.1, le cycle de fonctionnement de DIFAIRT débute avec la constitution de la base de Patterns. Son rôle est de stocker les modèles de questions sources sous des formats prédéfinis. Cette base est alimentée par l'enseignant lui-même, soit manuellement soit par l'importation de données existantes (respectant par exemple la norme IMS-QTI).

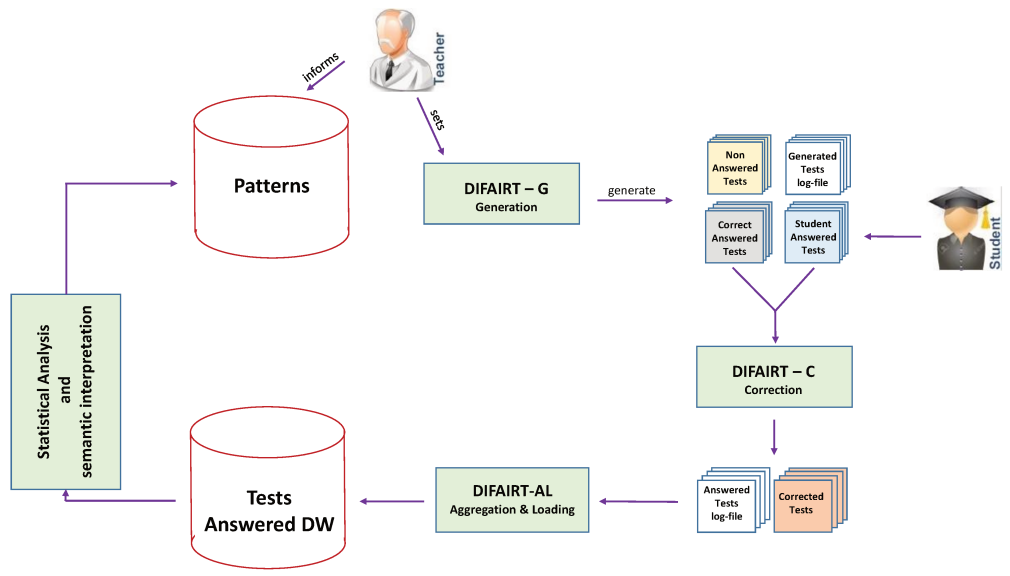

FIGURE 2.1 – Cycle général de fonctionnement du système

Le deuxième composant clé de DIFAIRT est le générateur de sujets différenciés appelé DIFAIRT-G. Se basant sur les contenus de la base et après le paramétrage de la génération par l'enseignant, on obtient une *Collection* (ensemble) de sujets différenciés appelés des *FairTests* ou encore des *Tests* (*FAIR*, pour dire qu'il s'agit de *Tests* équitables). On entend par paramétrage du générateur le fait de spécifier le nombre de Tests à générer (la taille de la *Collection* de *Tests* à générer) ; la quantité de questions dans chaque *Tests* et la quantité de choix de réponses par question. A terme, il est également prévu de spécifier des informations additionnelles comme le niveau de différenciation et de difficulté moyen entre les *Tests* à générer. A la génération, trois types de contenus sont générés par *Test*. Les premiers sont les *Tests* qui seront soumis à l'apprenant au moment de la composition, ils sont appelés *NonAnsweredTest* ; les deuxièmes sont les corrigés (*CorrectAnsweredTest*) qui seront utilisés au moment de la correction automatique des copies ; et le troisième type de fichier généré est l'historique de la génération, appelé *Log-File*.

Après la composition, le système récupère des *StudentAnsweredTest*. Ainsi, la base des *FairTests* contient trois types d'éléments par *Test* au final. Comme dans YMCQ, la correction automatique se fait par une comparaison automatique de chaque *StudentAnsweredTest* avec le *CorrectAnsweredTest*. Ce système de correction s'appelle DIFAIRT-C et dispose d'une base de données à part pour stocker les résultats de chaque *Test*. La méthode de correction suivie par DIFAIRT est calquée de celle du système YMCQ mais en apportant quelques améliorations par exemple YMCQ n'avait qu'un système de fichiers pour le stockage et ne conservait que les notes finales, contrairement à DIFAIRT qui à une base de

donnée dédiées à cela et d'avantage d'informations que les notes finales.

La dernière étape du cycle de DIFAIRT est gérée par le module DIFAIRT-AL. L'objectif de cette phase est d'abord de pouvoir faire des analyses statistiques sur les résultats. Par exemple, combien de fois l'*ItemTest* $a1$ a été tiré, combien de fois il a été répondu juste, combien de fois il a été répondu partiellement juste, combien de fois il a été répondu faux ou encore dans quelle configuration de tirage de choix il a été répondu juste, etc ? ? ? En complément de ces analyses statistiques ensuite est prévue une dimension sémantique pour éclairer certaines problématiques d'équité (en terme de niveau de difficulté dans les *Tests* différenciés). Ces interprétations statistiques et sémantiques ont pour but d'enrichir la base source d'informations complémentaires sur chaque pattern.

Toutefois, ces travaux de recherche se portent principalement sur les phases de conception et de génération de sujets d'évaluations.

2.3.2 Certains questionnements relatifs à ces travaux

Notre thèse pose la question de l'optimisation de la différenciation dans les *Tests* individuels auto-générés, en réduisant la taille de la base source. Pour les premiers travaux, nous avons fait le choix de travailler sur la différenciation dans les sujets de types QCM. Pour bien comprendre les problématiques pouvant découler de cela, il faut déjà comprendre la structure de base d'un sujet d'évaluation de type question à choix multiples, ce qui permettra de se rendre compte des premières difficultés.

Un *Test* QCM peut-être perçu comme une série d'éléments appelés des *ItemTests*. Chaque *ItemTest* est fait d'un ensemble de sous-Items qui sont un énoncé (*EnonceItem*) et des choix de réponses (*RepItemITs*). Se basant sur cette structure, on peut déjà dégager une première idée de la complexité de la différenciation. Parler de la différenciation suppose une manipulation intelligente des différents éléments et sous-éléments constituant un *Test* et ceci jusqu'à la granularité la plus petite à savoir les choix de réponse. Ainsi, la différenciation implique tous les éléments du *Test*, en leurs contenus et en leurs placements dans les *Tests* comparés. On entend par "en contenus", le fait de prendre en compte que chaque question dans un *Test* est ou n'est pas dans l'autre *Test*, et si elle y est, faire la même vérification dans les choix de réponse. On entend par "en placements", la vérification que lorsqu'un même contenu existe dans deux *Tests* comparés, vérifier qu'ils sont oui ou non dans la même position. Cette vérification est faite pour les questions, et pour les réponses. La différenciation permet de répondre aux questions suivantes : est-ce que l'énoncé Xi du *Test* j existe dans le *Test* i ? Si oui, sont-ils dans une même position sur les deux *Tests* ? Est-ce que les choix de réponses sont les mêmes dans les deux ? Si oui, sont-ils dans un même ordre (position) ? Pour ainsi dire, la différenciation implique à la fois les

énoncés et les choix. Toutefois, quand on parle d'équité, différents questionnements voire problématiques peuvent apparaître. Voici trois principaux questionnements :

- Le premier problème est lié à la structure de stockage et à l'analyse des résultats. En terme de stockage et d'analyse, puisque chaque sujet est différent, et que la différenciation implique aussi les choix de réponses, la quantité de donnée à stocker pour chaque épreuve devient considérable. Comment structurer ce stockage et que stocker pour faciliter la manipulation des données et avoir les meilleures analyses statistiques possibles ? De plus, on peut se poser la question du comment faire pour analyser les résultats des évaluations passées, afin de déduire des informations sur les questions et les réponses. Pour savoir par exemple, s'il s'agit de question facile ou difficile à répondre pour les apprenant qui avaient suivi la formation en question.

- Le deuxième est le problème d'équité dans des sujets différenciés. L'équité est considérée comme une contrainte à part dans le projet DIFAIRT. Comment un enseignant, voulant faire de la différenciation dans les sujets qu'il génère pour ses étudiants, peut-il s'assurer qu'ils sont d'un même niveau de difficulté ? Lorsque l'on s'intéresse à un sujet d'examen de type Questionnaire à Choix Multiples construit à partir d'une base de questions candidates, si le nombre de questions et le nombre de choix associés à chaque question dans la base source est strictement égal à ceux devant figurer sur chaque sujet, un simple tirage aléatoire dans l'ensemble des arrangements peut suffire à garantir à la fois la différenciation et un niveau acceptable d'équité (si l'on met de coté l'influence que peut avoir l'ordre d'une série de questions sur la perception de sa difficulté par un apprenant). Lorsque l'ensemble des questions candidates à la sélection est strictement plus grand que celui des questions figurant sur chaque sujet, garantir des niveaux de difficulté proches n'est plus si trivial. Si, pour une question donnée, seule les données d'entrée changent d'un sujet à l'autre, la solution peut rester relativement simple à mettre en œuvre. En revanche, être en mesure d'offrir les mêmes garanties alors que l'énoncé change est plus ardu.

- Le troisième problème est lié au fait que la différenciation dans les *Tests* (qu'ils soient générés manuellement ou automatiquement), implique une quantité de questions et de réponses importante. Par exemple un enseignant qui se retrouve avec 300 apprenants doit avant tout constituer une base de données proportionnellement conséquente, avant de prétendre concevoir 300 *Tests* tous différenciés deux à deux. Même en ayant à sa disposition cette base source assez conséquente, comment contrôler le niveau de différenciation en chaque couple de *Tests* et comment garantir l'évaluation de chaque apprenant avec un même niveau d'exigence ?

2.4 Problématiques et objectifs de nos travaux

Cette thèse se focalise plus spécifiquement sur la question de la génération automatique de *Tests* différenciés. Elle souhaite apporter des éléments de réponse aux problématiques liées à la différenciation dans les *Tests* et permettre d'assister l'enseignant dans l'élaboration de ses sujets dans des situations mentionnées dans la section 1.1.4 du chapitre 1. Ces travaux trouvent leur principal contexte scientifique dans le champ de recherche des EIAH, plus spécifiquement dans la thématique des Évaluations. On a vu qu'il existait peu de travaux de cette thématique qui questionnaient la notion de la différenciation dans les *Tests* d'évaluation. De ce fait, cette thèse aborde trois principaux problèmes qui sont :

- la génération de *Collections* de *Tests* (de type QCM pour l'instant) sous contrainte de la différenciation.
- l'optimisation de la contrainte de la différenciation moyenne dans toutes *Collections* de Tests auto-générées. Pour cela, il faudra pouvoir contrôler/mesurer cette différenciation, ce qui implique la conception et l'expérimentation d'une métrique permettant de le faire.
- des algorithmes de génération de *Collections* de *Tests* performants, permettant de maximiser la différenciation moyenne dans les *Collections* générées tout en minimisant le volume de données qui serait nécessaire à cela dans la base source de l'enseignant.

Notre premier objectif est d'être en mesure d'étudier et de proposer des possibilités pour garantir efficacement la différenciation dans des *Collections* auto-générées. Pour y arriver, nous comptons mettre en place une métrique spécifique, permettant de mesurer la différenciation entre deux *Tests*. Le but est d'avoir le contrôle de la différenciation entre chaque couple de sujets depuis sa génération. Ainsi, cette métrique doit pouvoir mesurer la différenciation entre tous les éléments constituants d'un *Test* (questions et choix de réponse) tout en tenant compte de la position de chacun de ces éléments dans les *Tests* comparés. Des méthodologies algorithmiques de génération de *Tests* différenciés doivent être étudiées et expérimentées, car avoir une métrique permettant le contrôle de la différenciation n'est pas suffisant sans des méthodes de génération adaptées. Ces travaux de recherche doivent également apporter des propositions d'assistance à la génération que l'on pourra proposer aux enseignants de façon à leur garantir les contraintes de différenciation souhaitées.

Dans le chapitre qui suit, nous faisons une présentation de notre façon d'aborder la contrainte de la différenciation des *Tests*, par la mise en place d'une métrique de mesure de différenciation entre les *Tests* auto-générés. Nous y présentons également une de nos premières approches de génération expérimentées dans la génération de *Collections* de *Tests* différenciées, de manière contrôlée.

Chapitre 3

Une métrique pour la différenciation dans les évaluations

Les précédents chapitres ont permis de comprendre que le principal objectif de cette thèse est de pouvoir générer des Collections de Test d'évaluation différenciés de types QCM, en faisant en sorte que la différenciation entre les Tests soit maximisée tout en minimisant la taille de la base source. Cela nécessite de contrôler la différenciation au cours du processus de génération. Ce chapitre vise à présenter *DTest()*, une métrique de mesure de la différenciation entre deux Tests d'évaluation auto-générés, qui constitue une première contribution de nos travaux de recherche. Une expérimentation de *DTest()* sur l'algorithme de référence pour la génération aléatoire de *Tests* issu de YMCQ est ensuite présentée. Cette expérimentation permet de tester et d'ajuster *DTest()* et à la fois de déterminer le périmètre des performances de cet algorithme à générer des patrons de *Collections* de *Test* différencié.

La première section détaille la structure d'un *Test* et présente la notation du projet DIFAIRT. La deuxième section introduit la métrique *DTest()* ; la troisième section est consacrée à la description de l'algorithme de génération de référence. *DTest()* est implémentée pour mesurer les performances en terme de différenciation ; une dernière section conclut en mettant l'emphase sur ce qu'il faut retenir des premiers résultats d'expérimentations.

3.1 Structure d'un Test et notation

DIFAIRT utilise son propre vocabulaire pour nommer ses principaux composants. Le tableau 3.1 ci-dessous synthétise le vocabulaire utilisé dans DIFAIRT pour identifier les composants d'un *Test*. Ainsi, pour DIFAIRT, un ensemble de *Tests* est appelé *Collection* et un *Test* est défini comme un ensemble de questions/éléments/Items appelés $ItemTest$.

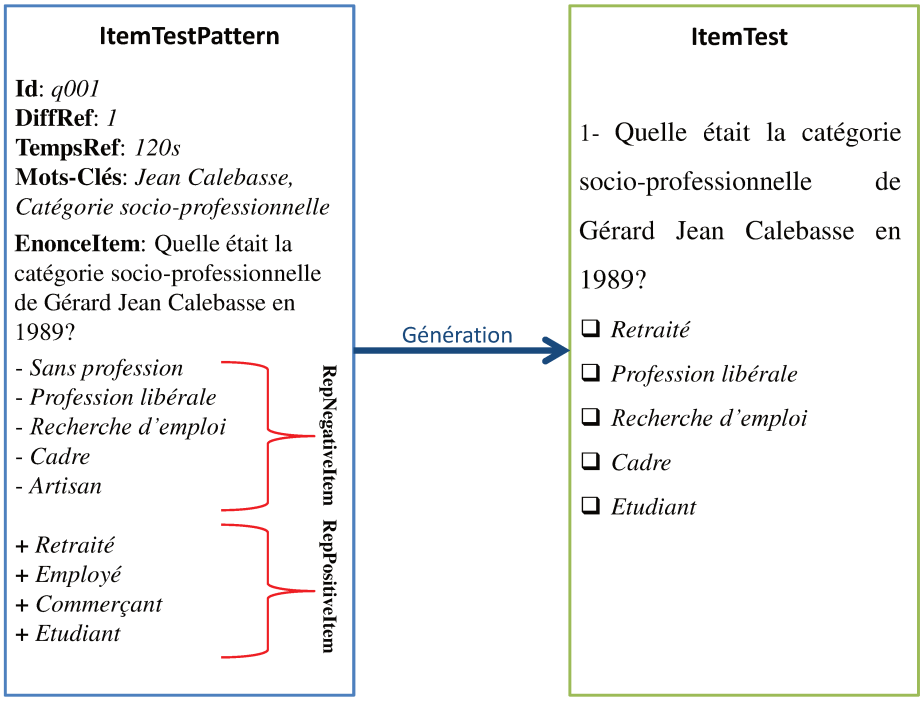

FIGURE 3.1 – Exemple de génération d'un *ItemTest* à partir d'un *ItemTestPattern*

Chaque $ItemTest$ est construit à partir d'un $ItemTestPattern$ provenant de la base source. Dans l'exemple présenté dans la figure 3.1, l'*ItemTestPattern* identifié au numéro "$q001$" est principalement fait d'un énoncé et d'un ensemble de choix de réponses. Les choix sont divisés en de bons et de mauvais choix de réponses qui sont respectivement précédées un signe plus ("+") et un signe moins ("−"). Dans 3.1, l'*ItemTest* "1" est donc créé à partir de l'*ItemTestPattern* "$q001$" en gardant le même énoncé et un sous-ensemble des choix de réponses. Ainsi, notre exemple comporte à gauche un *ItemTestPattern* avec un ensemble de choix de réponses constitué de 5 mauvaises réponses et de 4 réponses exactes. Le générateur constitue un *ItemTest* à droite composé de 5 choix de réponses dont 2 justes. Alors, un $ItemTestPattern$ est une question de la base source qui contient au moins un énoncé (appelé $EnonceItemITP$) et deux groupes de choix de réponses, dont le premier contient des $RepPositiveItemITP$ et le deuxième des $RepNegativeItemITP$ que l'on appelle aussi en pédagogie des distracteurs. Pour sa part, un *ItemTest* est fait d'un énoncé qui s'appelle $EnonceItemIT$ et d'un ensemble de choix de réponses, divisé en de bons et de mauvais choix appelés respectivement les $RepPositiveItemIT$ et les $RepNegativeItemIT$. Ainsi, les extensions *IT* sont pour faire référence à l'appartenance aux *ItemTests* (c'est-à-dire sur le sujet généré) et *ITP* pour faire référence à l'appartenance aux *ItemTestPatterns* (c'est-à-dire dans la base source).

Pour présenter les différents composants d'un *Test* plus formellement, nous utilisons

Termes	Définition
Collection	Un ensemble de Tests générés
ItemTestPattern	Questions de la base source (énoncés et choix de réponses)
ItemTest	Questions sur un Test généré (énoncés et choix de réponses)
EnonceItem	Énoncés des questions
EnonceItemIT	Énoncés dans un ItemTest
EnonceItemITP	Énoncés dans un ItemTestPattern
RepItem ou Ans	Choix de réponses des questions
RepItemIT	Choix de réponses dans un ItemTest
RepItemITP	Choix de réponses dans un ItemTestPattern
RepPositiveItemIT	Bons choix de réponses dans un ItemTest
RepNegativeItemIT	Mauvais choix de réponses dans un ItemTest
RepPositiveItemITP	Bons choix de réponses dans un ItemTestPattern
RepNegativeItemITP	Mauvais choix de réponses dans un ItemTestPattern

Tableau 3.1 – Table des noms des composants d'un *Test* DIFAIRT

certaines notations qui sont résumées dans le tableau 3.2. Il contient trois colonnes dont la première contient les notations, la deuxième contient les termes DIFAIRT et la dernière contient les définitions correspondant à chaque notation.

Notations	Termes	Définition
$R_{a_{IT}}$	RepPositiveItemIT	Choix de réponses correctes dans un ItemTest
$W_{a_{IT}}$	RepNegativeItemIT	Mauvais choix de réponses dans un ItemTest
$R_{a_{ITP}}$	RepPositiveItemITP	Bons choix de réponses dans un ItemTestPattern
$W_{a_{ITP}}$	RepNegativeItemITP	Mauvais choix de réponses dans un ItemTestPattern
ans_i	RepItemIT	Quantité de choix de réponses totale disponible dans un ItemTest i

Tableau 3.2 – Table de notation formelle de DIFAIRT (plus de détails dans le glossaire en annexe)

Ainsi, un $ItemTestPattern$ est fait d'un énoncé, de $\#R_{a_{ITP}}$ bons choix de réponses pour ($\#R_{a_{ITP}} \geq 1$), et de $\#W_{a_{ITP}}$ de mauvais choix de réponses pour ($\#W_{a_{ITP}} \geq 1$). Ainsi, un *ItemTestPattern j* peut être présenté comme suit :

- *EnonceItemITP* : son énoncé
- $R_{a_{ITP_j}}$, $j \in \{1,\ldots,\#R_{a_{ITP}}\}$: ses bons choix de réponses
- $W_{a_{ITP_j}}$, $j \in \{1,\ldots,\#R_{a_{ITP}}\}$: ses mauvais choix de réponses

L'*ItemTest* contient un énoncé, un sous-ensemble de $\#R_{a_{IT}}$ de bons choix de réponses, et de $\#W_{a_{IT}}$ mauvais choix de réponses. Un *ItemTest i* peut être défini ainsi :

- *EnonceItemIT* : son énoncé
- ans_i=$\{R_{a_{IT_i}}, i = 1,\ldots,\#R_{a_{IT}}(\#R_{a_{IT}} \leq \#R_{a_{ITP}})\} \cup \{Wa_i, i = 1,\ldots,\#W_{a_{IT}}($ $\#W_{a_{IT}} \leq \#W_{a_{ITP}})\}$: c'est-à-dire, l'ensemble des choix de réponses justes ou faux provenant de ceux de l'*ItemTestPattern* correspondant.

Lors de la génération, un *ItemTestPattern* peut donner naissance à plusieurs *ItemTests* différents. Dans notre cas, des *ItemTests* provenant d'un même *ItemTestPattern* auront un même énoncé mais des choix de réponses différents ou placés dans des positions différentes. Un *ItemTest* x généré à partir d'un *ItemTestPattern* y peut se définir comme suit :

- $ItemTest_x.EnonceItemIT = ItemTestPattern_y.EnonceItemITP$
- $ItemTest_x.ans_i = \{a \in ItemTestPattern_y.r_a\} \cup \{rep_i \in ItemTestPattern_y.r_a \backslash \{a\} \cup ItemTestPattern_y.w_a\}$

Ainsi, lors de la construction des choix de réponses dans l'*ItemTest* x, une bonne réponse est d'abord tirée et les autres sont tirées aléatoirement parmi les bonnes et mauvaises réponses. Ce procédé garantit le tirage d'au moins une bonne réponse pour chaque *ItemTest* créé. La construction de chaque *Test* d'une collection est faite en suivant les trois principales conditions suivantes :

- $\#R_{a_{IT}} + \#W_{a_{IT}} \leq \#R_{a_{ITP}} + \#W_{a_{ITP}}$: les choix de chaque *ItemTest* est inférieur ou égal aux choix présents dans l'*ItemTestPattern* à partir duquel il est créé.
- $\#R_{a_{IT}} + \#W_{a_{IT}}$: représente la quantité de choix de réponse par *ItemTest* et elle est fixée par l'enseignant au moment du paramétrage de la génération.
- $p \leq n$ (p représente la quantité d'*ItemTests* présents dans un *Test* et n la quantité *ItemTestPatterns* présents dans la base. Alors, la quantité d'éléments constituants du *Test* ne peut être supérieure à ceux présents dans la base source).

3.2 DTest(), la métrique proposée et complexité

La notation étant établie et la structure des *Tests* explicitée, cette section est consacrée à la métrique que nous proposons pour le calcul de la différenciation entre les *Tests*. Parler d'une métrique pour le calcul de la distance entre deux *Tests* suppose la prise en compte de la combinatoire engendrée par le contenu et la structure d'un *Test* qui dans notre cas est un QCM. Plus précisément, cette métrique doit pouvoir dire quel est le niveau de différenciation qui existe entre deux *Tests* en se basant sur les énoncés de questions et les choix de réponses qui les constituent.

Pour DIFAIRT, un *Test* se présente comme une série d'*ItemTests*. En le présentant ainsi, on peut dire que notre métrique a pour objectif de mesurer la distance structurelle entre deux séries d'éléments (deux séries d'*ItemTests*). Par contre, on sait qu'il s'agit d'éléments complexes puisque chaque *ItemTest* est caractérisé dans un *Test* par son placement, et son contenu. On entend par son placement, sa position dans le *Test* et par son contenu, les sous-éléments qui le compose (i.e l'énoncé et les choix de réponses).

La première difficulté consiste en la prise en compte à la fois des questions et des réponses dans chaque *Test*. C'est-à-dire, qu'il faut être capable de considérer la différence de position entre les *ItemTests* et les choix de réponses. Pour cela, la métrique doit questionner la distance structurelle globale entre deux *Tests*. La question posée est alors : considérant l'un des *Tests* comme référence, à quelle "distance" se trouve-t-il du second (i.e. de la "cible") ? Cette distance globale est alors une somme de distances locales considérant chaque *ItemTest* du *Test* de référence. Ainsi, un *ItemTest* de référence n'apparaissant pas dans le *Test* cible est considéré comme étant de distance maximale. On dit dans ce cas qu'il est disparate. A l'inverse, un *ItemTest* de référence apparaissant dans le *Test* cible, de surcroît au même numéro d'ordre, illustre une distance minimale. La gradation entre ces deux situations extrêmes fait intervenir à la fois la différence du numéro d'ordre de l'*ItemTest* dans la cible mais aussi la structure de l'ensemble des *RepItemITs*.

Le deuxième niveau de difficulté est de faire en sorte que la métrique soit pédagogiquement sensée. C'est-à-dire, retourner la valeur minimale pour deux *Tests* complètement identiques (même contenus dans les mêmes placements), et la valeur maximale entre deux *Tests* complètement disparates (formés d'éléments complètement différents). Également, elle doit être pédagogiquement sensée dans les cas intermédiaires, donc de retourner des valeurs significatives pour des *Tests* plus ou moins semblables dans leurs questions, dans leurs réponses et les placements de ces derniers.

La suite de cette section présente le coté technique de cette métrique et son expérimentation.

3.2.1 Inspirations et principes de la métrique

Si l'on associe un identifiant unique à chaque *ItemTest*, un *Test* (étant une suite ordonnée d'*ItemTests*) s'apparente, à un "mot" construit sur l'alphabet des identifiants. Adoptant ce point de vue, chaque *EnonceItem* d'un *ItemTest* représente un symbole appartenant à notre alphabet. Ainsi, la distance entre deux "mots" s'apparente à la distance entre deux *Tests* dans notre cas. Ce choix conceptuel nous permet d'adapter une partie de ces travaux à une communauté d'algorithme qui est l'algorithmique du texte. Dans ce contexte, différentes approches de calcul de distances entre chaînes de caractères ont ainsi été étudiées de façon à déterminer celles qui pourraient contribuer à répondre à notre problématique.

La distance de Hamming [52] permet de mesurer la différence entre deux chaînes de caractères de même longueur. Cette distance est reprise dans divers travaux de recherche en informatique et télécommunication. Elle peut se définir comme suit : soient deux mots M_1 et M_2 de longueurs respectives $l1$ et $l2$ où ($l1 = l2$), créés à partir d'un alphabet

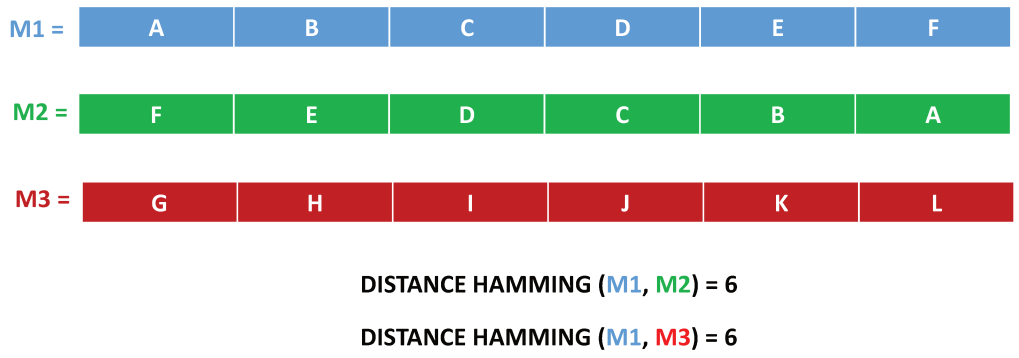

FIGURE 3.2 – Exemple de la distance Hamming

A, la distance de Hamming entre M_1 et M_2 est le nombre de symboles à des positions identiques pour lesquels M_1 et M_2 diffèrent. Donc,

$$Hamming(M_1, M_2) = \sum_{k=1}^{n} \begin{cases} 1 & si(M_{1_k} = M_{2_k}) \\ 0 & si(M_{1_k} ! = M_{2_k}) \end{cases}$$

pour $n = l1 = l2$

Ainsi, lors de la comparaison de deux chaînes, la distance de Hamming retourne zéro (0) lorsqu'elles sont identiques. Sinon elle renvoie la valeur représentant le nombre de différences dans les deux chaînes de caractères. Le changement de place d'un caractère et la non-existence d'un caractère d'une chaîne à une autre ont le même poids. Dans le cas des *Tests* comme la figure 3.2 le montre, cela voudrait dire que la distance entre deux *Tests* ayant les mêmes contenus dans des positions différentes comme ($M1$ et $M2$), serait équivalente à deux *Tests* ayant des contenus complètement disparates comme ($M1$ et $M3$). Du point de vue pédagogique ceci n'est pas forcément réaliste. En effet, deux sujets ayant les mêmes questions et réponses dans des placements différents sont moins différenciés que deux sujets ayant des contenus complètement différents.

La distance de Levenshtein [41] quant à elle, permet d'estimer la distance entre deux chaînes de caractères en calculant le nombre d'insertions, de suppressions et de substitutions de caractères qu'il faut réaliser dans une des chaînes pour qu'elle devienne identique à l'autre [78]. Souvent considérée comme une extension voire une généralisation de la distance de Hamming, cette distance permet de prendre en compte plus de caractéristiques dans les chaînes qu'elle compare. La distance de Levenshtein permet d'allouer un poids, au fait que deux éléments identiques changent de place d'une chaîne à une autre ; un autre poids au fait qu'un élément soit supprimé dans une des chaînes ; et un troisième

poids au fait qu'un nouvel élément soit inséré dans une des chaînes. Elle est retrouvée dans plusieurs domaines comme la reconnaissance vocale et la reconnaissance des formes [23] [76]. A noter de plus que cette méthode permet de calculer la distance entre deux chaînes de longueurs différentes.

Plus formellement, elle peut se définir comme suit : soient deux mots M_1 et M_2 de longueurs respectives $l1$ et $l2$, créés à partir d'un alphabet A, la distance de Levenshtein entre M_1 et M_2 est le nombre minimal d'opérations d'édition pour transformer M_1 en M_2. Elle sera comprise entre zéro (0) et la longueur de la chaîne la plus longue. Supposant que $M_1 = "ABA"$ et $M_2 = "CAC"$, la figure 3.3 présente une simulation du calcul de la distance de Levenshtein entre M_1 et M_2. Cette distance est de 3 car il faut faire au moins trois opérations dans la chaîne M_1 pour qu'elle devienne identique à la chaîne M_2. En suivant le déroulement des opérations dans l'exemple de la figure 3.3, on voit que la première opération est la suppression du caractère "A", la deuxième est la substitution de "B" par "C", et la dernière opération une insertion du caractère "C". Techniquement, à partir des deux mots M_1, M_2, on définit la table T à $l1+1$ lignes et $l2+1$ colonnes par $T[i,j] = Lev(M_1[0..i], M_1[0..j])$ pour $(i = -1, 0, ..., l1-1)$ et $(j = -1, 0, ..., l2-1)$. Pour calculer $T[i,j]$, on utilise la formule de récurrence suivante : Pour $(i = 0, 1, ..., l1-1)$ et $(j = 0, 1, ..., l2-1)$, on a $T[-1,-1] = 0$; $T[i,-1] = T[i-1,-1]+1$; et $T[-1,j] = T[-1,j-1]+1$.

$$T[i,j] = \min \begin{cases} T[i-1,j-1] + Sub(x[i],y[j]) & si(M_{1_k} = M_{2_k}) \\ T[i-1,j]+1 & si(M_{1_k} ! = M_{2_k}) \\ T[i,j-1]+1 & si(M_{1_k} ! = M_{2_k}) \end{cases}$$

où

$$Sub(a,b) = \begin{cases} 0 & si(a=b) \\ 1 & sinon \end{cases}$$

Nous avons décidé de nous inspirer de la distance de Levenshtein pour créer la distance *DTest()*. Le tableau 3.4 présente une analogie empirique entre ce que propose Levenshtein et *DTest()*. Ainsi, pour mesurer la distance entre deux *Tests*, nous proposons de mesurer le nombre d'*ItemTests* qui comportent le même énoncé, et en cas de similitude leur différence de position d'un *Test* à l'autre est calculée. Pour cela, *DTest()* se base sur les notions de disparité et de permutation entre les éléments constituants les *Tests*. Il existe une disparité entre deux *ItemTests* si les énoncés sont différents d'un *Test* à l'autre. Il existe une permutation entre deux *ItemTests* si les énoncés sont les mêmes dans les deux *Tests*, mais situés à des emplacements différents.

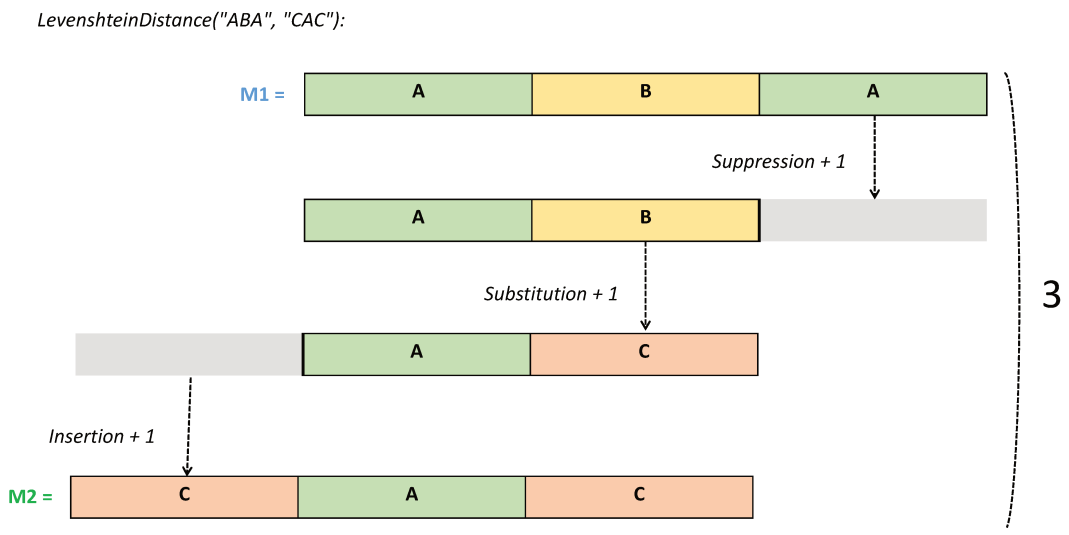

FIGURE 3.3 – Simulation distance de Levenshtein sur les chaines $M_1 = "ABA"$ et $M_2 = "CAC"$

Définitions	
Distance égale au nombre minimal d'opérations de suppressions, d'insertions et de substitutions qu'il faut faire dans une chaîne de caractère pour qu'elle devienne identique à l'autre	Distance égale au nombre minimal d'opérations de suppressions, d'insertions et de substitutions qu'il faut faire dans un Test pour qu'il devienne identique à l'autre
Levenshtein	**DTest**
Calcul de la distance entre deux « mots» (suites de caractères)	Calcul de la distance entre deux «mots» (suites d'ItemTests et de RepItemITs)
Principales Operations	
Substitution d'un caractère	Substitution d'un ItemTest (ou d'un RelItemIT)
Suppression d'un caractère	Suppression d'un ItemTest (ou d'un RelItemIT)
Insertion d'un caractère	Insertion d'un ItemTest (ou d'un RelItemIT)

FIGURE 3.4 – Première analogie entre Levenshtein et *DTest*

De plus, puisque un *Test* comporte une structure bien plus complexe qu'une simple succession d'*ItemTests* représentés par des identifiants uniques, notre distance doit tenir compte des ensembles de choix de réponses associé à un énoncé. *DTest()* prend également en compte la disparité des choix de réponses associés aux énoncés d'*ItemTests*, et/ou leurs permutations. En effet, deux *EnonceItemITs* peuvent être identiques, alors que l'ensemble de leurs choix de réponses possibles *RepNegativeItems*, *RepPositiveItems* est différent et/ou organisé dans un ordre différent.

Pour construire *DTest()*, il nous a fallu tout d'abord déterminer pédagogiquement quels étaient les critères permettant de mesurer la proximité ou le niveau de ressemblance entre deux *Tests*. Ainsi, si l'on considère une échelle de distance structurelle bornée sur l'intervalle [0, 1], nos critères de différenciation nous amènent à identifier quatre plages de valeurs illustrées dans la figure 3.5. Suivant cette échelle, la valeur minimale de *DTest()*, zéro (0) est atteinte lorsque les deux *Tests* comparés sont identiques, mêmes *ItemTests* avec *EnonceItemITs* et *RepItemITs* aux mêmes emplacements. A l'opposé, sa valeur maximale est un (1) lorsque les *Tests* sont totalement disparates, c'est-à-dire s'ils n'ont aucun *ItemTest* en communs (et par conséquent aucun *RepItemITs* en communs). Pour les situations intermédiaires, notre échelle de distance propose une gradation suivant le zonage suivant :

- *cp* (choice permutation) : représentant la zone où les deux *Tests* ont les mêmes *ItemTests*, dans un ordre faiblement différent avec des *RepItemITs* faiblement différents si l'on se réfère aux éléments définis par la distance de Levenshtein. la zone *cp* identifie donc l'intervalle de faible permutation des *ItemTests* et faible disparité des *RepItemITs*.
- *qp* (question permutation) : représentant la zone où les deux *Tests* ont des *ItemTests* identiques, mais dans un ordre fortement différent avec des *RepItemITs* faiblement différents si l'on se réfère aux éléments définis par la distance de Levenshtein. *qp* est l'intervalle de forte permutation des *ItemTests* et faible disparité des *RepItems*.
- *cd* (choice disparity) : les deux *Tests* ont quasiment les mêmes *ItemTests* où les *RepItemITs* sont disparates si l'on se réfère aux éléments définis par la distance de Levenshtein. C'est l'intervalle de forte disparité des *RepItemITs*.
- *qd* (question disparity) : les *ItemTests* sont disparates si l'on se réfère aux éléments définis par la distance de Levenshtein. C'est l'intervalle de forte disparité des *ItemTests*

Alors, pour deux *Tests* $T1$ et $T2$, la fonction *DTest()* fait appel à quatre sous-fonctions correspondant aux quatre zones présentées dans le schéma 3.5 pour : le calcul de la disparité entre les *ItemTests* (*qd* : question disparity) ; le calcul de la permutation entre les *ItemTests* communs (*qp* : question permutation) ; le calcul de la disparité entre les choix

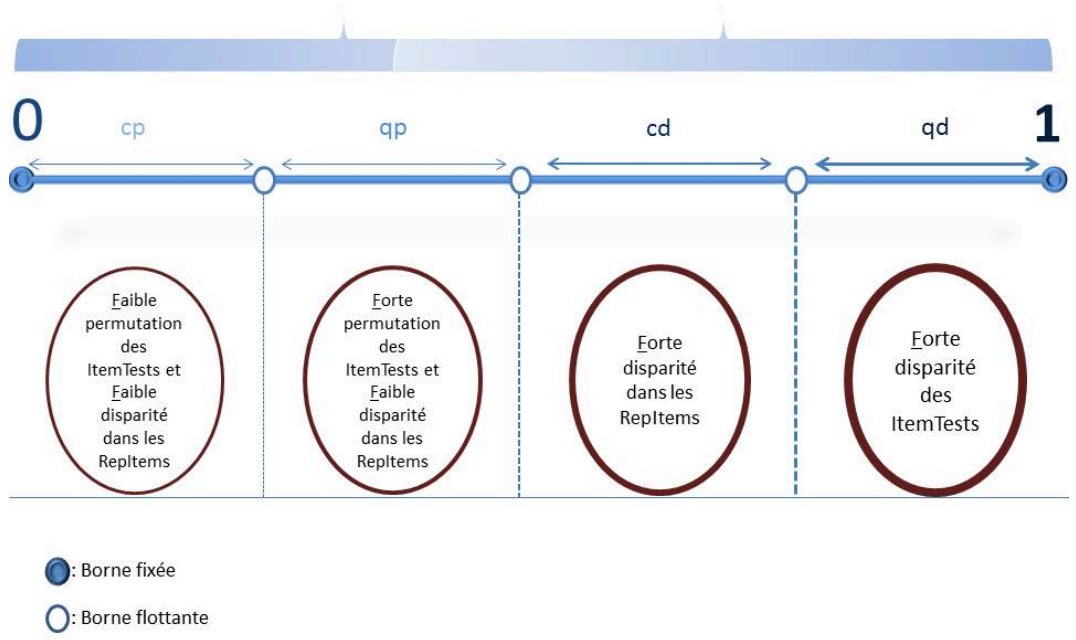

FIGURE 3.5 – Schéma de zonage de DTest()

des *ItemTests* communs (*cd* : choice disparity) ; et le calcul de la permutation entre les choix des *ItemTests* communs, contenant des choix communs (*cp* : choice permutation). Ces quatre zones sont ordonnées par la différenciation pédagogique, c'est-à-dire pour nous les *Tests* comparés sont moins proches en *cd* qu'en *qp*, ainsi de suite.

L'expression finale de *DTest()* repose sur deux fonctions. La première fonction (3.1) permet de calculer la disparité entre deux séries d'éléments. Dans notre cas, ces éléments sont les questions présentes dans un *Test* ou les choix de réponse des *ItemTests*. Soient a et b, deux séries d'éléments quelconques définies sur un alphabet A, $Disparity(a,b)$ retourne la quantité d'éléments disparates entre elles. Cette fonction peut se représenter par la formule 3.1.

$$Disparity(a,b) = \frac{max(\#a,\#b) - \#(a \cap b)}{max(\#a,\#b)} \tag{3.1}$$

La seconde fonction (3.2) permet de calculer la différence de placement entre deux éléments communs dans deux séries d'éléments ($a \cap b$) soient a et b deux mots définis sur un même alphabet A. Dans notre cas, ces éléments sont des questions communes dans les deux *Tests* comparés ou les réponses communes dans les questions communes. Donc, $Permutation(a,b)$ retourne la différence de places entre les éléments communs dans a et b. Cette fonction peut se représenter par la formule 3.2 .

$$Permutation(a,b) = \sum\nolimits_{i \in (a \cap b)} \mid x_i^a - x_i^b \mid \tag{3.2}$$

La nécessité de normaliser cette expression suppose qu'on puisse la diviser par sa valeur maximale donnée par : $2\left\lfloor \frac{max(\#a,\#b)}{2} \right\rfloor \left\lceil \frac{max(\#a,\#b)}{2} \right\rceil$

La valeur finale de *DTest()* est normalisée, donc entre 0 et 1. Pour que la valeur de distance retournée par *DTest()* soit pédagogiquement sensée (respecter l'échelle de la figure 3.5), des constantes de valeur (poids), nommées *left-percent-qd*, *left-percent-cd* et *left-percent-qp* sont respectivement allouées à *qd*, *cd* et *qp* pour fixer leur poids d'importance dans la différenciation des *Tests* comparés. C'est-à-dire, *qd* a plus de poids que *cd* qui lui a plus de poids que *qp* et qui ont plus de poids que *cp*. Cette fonction finale de *DTest()* peut se représenter par l'algorithme 1 qui montre qu'après le calcul des quatre valeurs de *DTest* (qd, cd, qp et cp), les constantes de valeurs (*left-percent-qd*, *left-percent-cd* et *left-percent-qp*) sont manipulées d'une façon permettant que la valeur de *qd* ait plus d'importance que *cd* qui à son tour aura plus de valeur que *qp*, ainsi de suite jusqu'à *cp* qui a le moins d'importance dans la valeur finale de *DTest()*. Les résultats de ces suites d'opérations sont gardés dans les variables $borne_max$, $resultat$ et $bound$. La valeur finale de *DTest()* se trouve à la fin du processus dans la variable $resultat$.

Algorithm 1 DTest()

Require:

$borne_max$: {variable tampon}
$resultat$: {variable qui contient le résultat final}
$borne$: $left_percent_qd$: {constante de valeur pour qd}
$left_percent_cd$: {constante de valeur pour cd}
$left_percent_qp$: {constante de valeur pour qp}
qd, cd, qp, cp : {quatre valeurs de l'échelle 3.5}
$t1, t1$: {deux Tests}

Ensure:

$qd \leftarrow questionDisparity(t1, t2)$: {calcul de la disparité entre les énoncés des questions}
$cd \leftarrow choiceDisparity(t1, t2)$: {calcul de la disparité entre les choix des questions communes}
$qp \leftarrow questionPermutation(t1, t2)$: {calcul de la permutation entre les énoncés communs}
$cp \leftarrow choicePermutation(t1, t2)$: {calcul de la permutation entre les choix des questions communes}
{Pour qd}
    $borne_max \leftarrow 1$
    $resultat \leftarrow qd$
    $bound \leftarrow (borne_max - resultat) * left_percent_qd$
{Pour cd}
    $borne_max \leftarrow bound + resultat$
    $resultat \leftarrow resultat + (cd * bound)$
    $bound \leftarrow (borne_max - resultat) * left_percent_cd$
{Pour qp}
    $borne_max \leftarrow bound + resultat$
    $resultat \leftarrow resultat + (qp * bound)$
    $bound \leftarrow (borne_max - resultat) * left_percent_qp$
{Pour cp}
    $resultat \leftarrow resultat + (cp * bound)$
return $resultat$

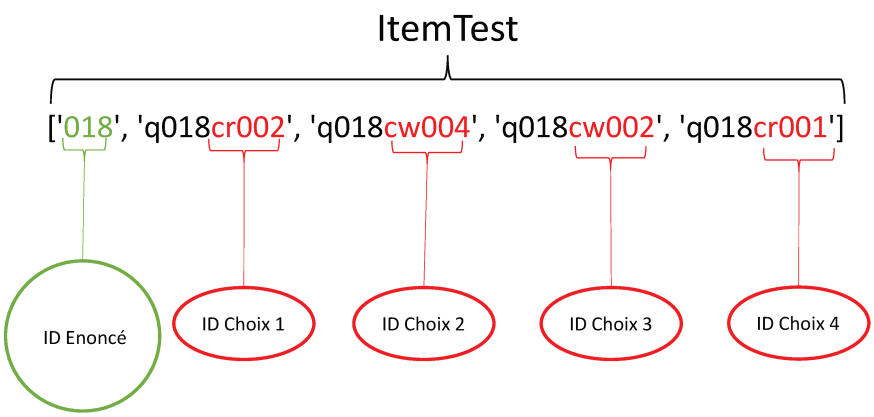

FIGURE 3.6 – Exemple d'identifiant d'un *ItemTest*

La sous-section suivante présente l'implantation, l'expérimentation et la validation de DTest().

3.2.2 Détermination des constantes de valeur de *DTest()*

La fonction principale de *DTest()* permet de prendre en paramètre les deux *Tests* à comparer sous un format prédéfini. Ce format est une chaîne d'éléments représentant un identifiant d'*ItemTest*. A travers l'identifiant de l'*ItemTest*, on retrouve ceux des *RepItemITs* qu'il renferme. Par exemple, la figure 3.6 présente un *ItemTest "['018', 'q018cr002', 'q018cw004', 'q018cw002', 'q018cr001']"*. L'identifiant de l'énoncé est *"018"* et les choix de réponses sont : *"q018cw004"*, *"q018cw002"* et *"q018cr001"*.

L'enjeu principal de cette expérimentation est de fixer les trois variables d'importance pour que les résultats respectent les quatre zones de la figure 3.5. Ainsi, nous avons formaté les variables *left-percent-qd*, *left-percent-cd* et *left-percent-qp* en pourcentages (allant de 0% à 100%). Pour trouver la bonne valeur de chacune de ces variables, on les a empiriquement fait varier sur tous les pourcentages possibles en calculant à chaque fois la distance entre des couples de *Tests* dont on connaît à l'avance leur niveau de différenciation, afin d'avoir une uniformité dans les distances. Ces *Tests* sont illustrés dans les figures 3.7, 3.8, 3.9 et 3.10.

En validation, les résultats respectent le zonage de DTests() pour *left-percent-qd* fixée à 70% (0.7), *left-percent-cd* à 50% (0.5) et *left-percent-qp* à 90% (0.9). En d'autres termes, une fois les valeurs de la disparité entre les questions *(qd)*, la disparité entre les choix *(cd)*, la

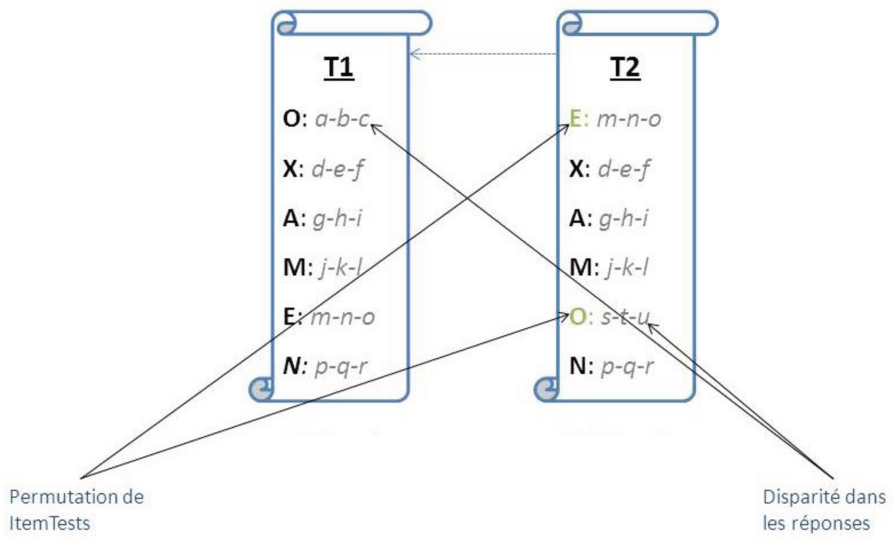

FIGURE 3.7 – Validation du cas de faible permutation dans les énoncés et de faible disparité dans les réponses (DTest proche de 0)

permutation entre les questions *(qp)* et la permutation entre les choix *(cp)* sont calculées, *qd* est multiplié par 0.7, *cd* par 0.5 et *qp* par 0.9. La différenciation globale entre deux Tests est alors une combinaison des quatre valeurs pondérées.

Quatre cas significatifs sont considérés pour vérifier la capacité de *DTest()* à calculer la distance entre des sujets différents . Ces cas sont illustrés à travers les figures 3.7, 3.8, 3.9 et 3.10. Notamment, la figure 3.7 représente deux *Tests* quasi identiques avec une faible permutation d'éléments. En observant de près cette figure, on peut voir que la différence entre eux réside dans la permutation de seulement deux énoncés (E et O). Donc, *DTest()* a retourné une distance proche de zéro (0).

Le deuxième cas représenté dans la figure 3.8 est celui où il y aurait deux *Tests* quasi identiques en terme de contenu avec beaucoup de permutations dans les questions et une faible disparité dans les choix. Donc, *DTest()* a retourné une distance entre eux proche de zéro (0) mais très supérieure par rapport au premier cas.

Le troisième cas représenté dans la figure 3.9 est celui où il y aurait deux *Tests* quasi identiques en énoncé mais très différencié en choix. Donc, *DTest()* a retourné une distance entre eux proche assez de un (1).

Le quatrième cas représenté dans la figure 3.10 est celui où il y aurait deux *Tests* disparates dans les énoncés. Donc, *DTest()* a retourné une distance entre eux est encore

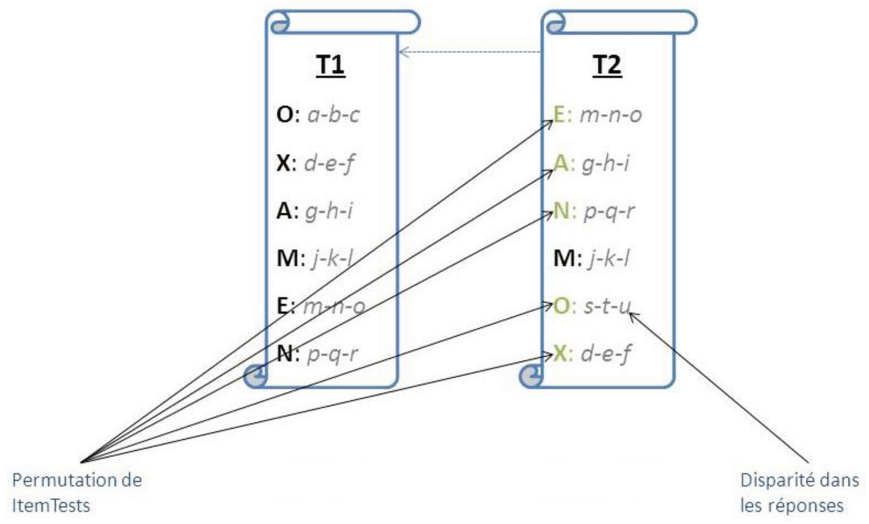

FIGURE 3.8 – Validation du cas de forte permutation dans les énoncés et de faible disparité dans les réponses (DTest proche de 0.5)

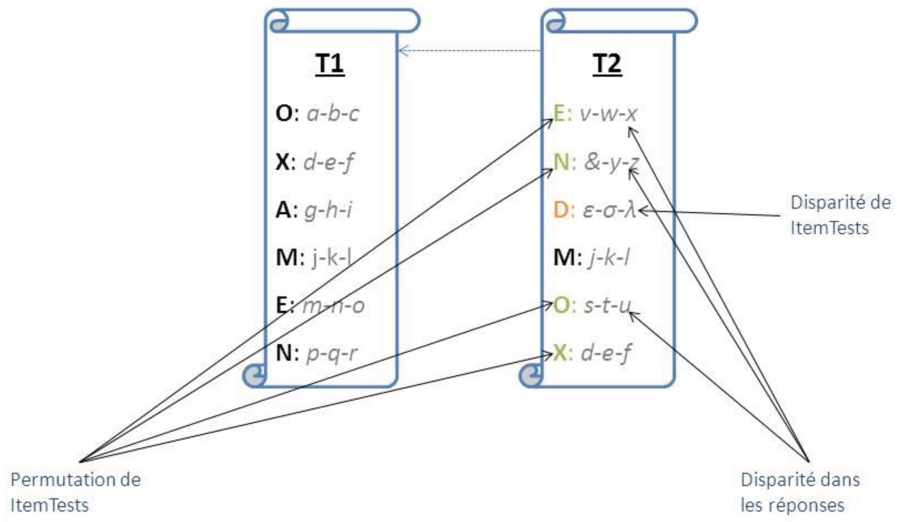

FIGURE 3.9 – Validation du cas de forte disparité dans les réponses (DTest proche de 0.7)

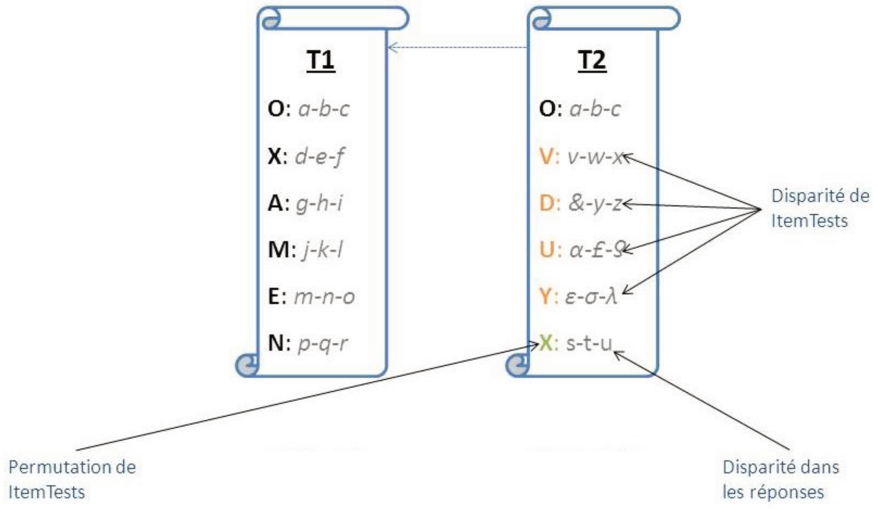

FIGURE 3.10 – Validation du cas de forte disparité dans les énoncés (DTest proche de 1)

plus proche de un (1) par rapport au troisième cas.

Pour les autres configurations possibles en terme de permutations et disparités des contenus, des expérimentations additionnelles ont montrées que *DTest()* retournait toujours des distances respectant le zonage défini. Dans la section suivante, un algorithme de génération de Collections de Tests basé sur une approche aléatoire est mis en place et sa capacité à générer des sujets différenciés est mesurée grâce à *DTest()*.

3.3 La métrique *DTest()* face à la génération aléatoire

Cette section est dédiée à la présentation d'une méthode de génération de *Tests* d'évaluation différenciés basée sur une approche aléatoire. *DTest()* est utilisée pour mesurer la performance de ce dernier en terme de niveau de différenciation entre les *Collections* de *Tests* qu'il génère. Cela nous a également donné des résultats de référence pour les autres algorithmes de génération. Les sous-sections suivantes présentent le fonctionnement, les composants clés de l'algorithme de génération, ainsi que les performances de cette méthode en terme de différenciation.

3.3.1 Composants et complexité de l'algorithme aléatoire

Pour la génération de chaque *Collection* de *Tests*, l'enseignant procède d'abord à la configuration du générateur en fixant la taille de la *Collection* ; la quantité de questions par *Test* ; et la quantité de réponses par question. Grâce à la base d'*ItemTestPatterns*, le

générateur peut construire une *Collection* de *m Tests* différenciés en procédant à un tirage aléatoire sans remise de *p ItemTestPatterns*. L'ordre des tirages correspond à l'ordre du placement des éléments dans les *Tests*. L'algorithme de génération aléatoire est présenté dans l'algorithme 2 (soient $randomPullItemTest()$ et $randomPullRepItem()$ deux fonctions permettant respectivement d'effectuer des tirages aléatoires sans remise d'*ItemTests* et de *RepItemITs* dans la base source).

Algorithm 2 Génération Aléatoire : randomGenerateTest()

Require:
  db : {base source contenant les ItemTestPatterns}
  $nbOfItemTests$: {nombre d'ItemTests par Test}
  $nbOfRepItems$: {nombre de RepItems par ItemTest}
  $nbOfTests$: {nombre de Tests par Collection}
  c : {une collection}
  $test$: {un Test}
Ensure:
  while $\#c \leq nbOfTests$ **do**
    repeat
      $ItemTest \leftarrow randomPullItemTest(db)$
      $RepItem \leftarrow randomPullRepItem(db, ItemTest, nbOfRepItems)$
      $test \leftarrow test \cup \{ItemTest + RepItem\}$
    until $\#test = nbOfItemTests$
    $c \leftarrow c \cup \{test\}$
  end while
  return c

Ainsi, pour générer un *Test* de *p ItemTests*, le générateur fait une sélection de façon aléatoire dans la base de *n ItemTestPatterns*. Pour construire un $ItemTest_i$ avec x choix de réponses, l'algorithme s'appuie sur un $ItemTestPattern_j$, en extrait d'une part l'$EnonceItem_j$ correspondant, puis tire parmi les $RepItemITP_j$ possibles, un sous-ensemble de x éléments correspondant aux choix qui vont alimenter l'$ItemTest_i$.

Le tirage des choix de réponses suit la même méthode de tirage aléatoire sans remise. Plus concrètement, un premier choix ans_i est tiré aléatoirement sans remise parmi les ra_{ITP_i} (donc parmi les bons choix réponses). Le premier tirage est fait ainsi pour garantir que parmi les choix qui seront tirés pour l'$ItemTest_i$, il y aura au minimum un bon choix (3.11). Ensuite, un autre tirage aléatoire sans remise de $x-1$ éléments est fait pour compléter les choix liés à la question parmi les $\{wa_{ITP_i} \cup ra_{ITP_i} \backslash ans_i\}$. Ainsi, le générateur fait un mélange aléatoire sur l'ensemble des QCM grâce à l'algorithme de Fisher-Yates qui a été popularisé par Knuth [8].

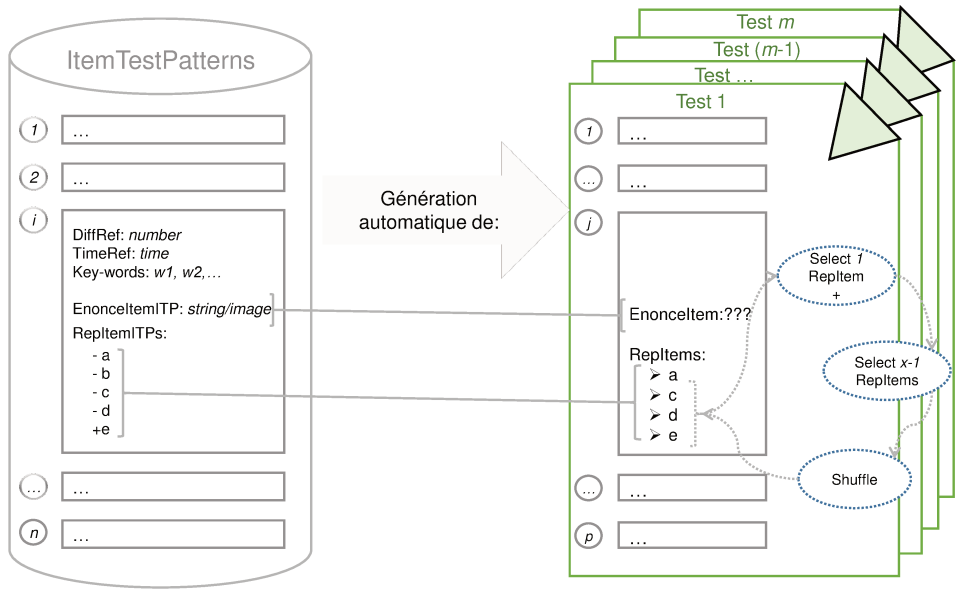

FIGURE 3.11 – Cycle de génération d'une collection d'une collection de tests

C'est à partir de cette méthode de génération que se pose le problème : *[Comment s'assurer que chaque Test d'évaluation est différencié des autres ?]*. Ceci est conditionné par les exigences minimales mentionnées ci-dessus (c'est-à-dire $p \leq n$). Ainsi, à partir d'échantillons aléatoires successifs de p *ItemTests* parmi les n *ItemTestPatterns* disponibles, la taille de l'ensemble de *ItemTests* que vous pourrez générer correspond à une p *permutation* de n (noté ${}_nP_p$) et égale à $\frac{n!}{(n-p)!}$.

Pour chacun de ces $ItemTests_i$, il existe $\left|ra_{IT_i}\right|$ possibilités de tirage $RepPositiveItemIT$. Ainsi, le nombre d'ensembles de choix multiples ordonnés, à l'exclusion du premier $RepPositiveItemIT$, est alors égal à une permutation d'éléments $x-1$ entre $\left|ra_{IT_i IT_i}\right| - 1$ et ainsi ${}_{IT_i \cup wa_{IT_i}|-1}P_{x-1}$. Enfin, en suivant la séquence décrite précédemment, cela signifie que ce générateur consiste en un tirage aléatoire d'un *Test* dans l'ensemble des *Tests* possibles et de cardinalité :

$$ {}_nP_p \,.\, \left|ra_{IT_i}\right| \,.\, {}_{|ra_{IT_i} \cup wa_{IT_i}|-1}P_{x-1} \quad (1 \leq i \leq p) \tag{3.3} $$

Il est intéressant de noter que pour un cas particulier, qui consiste exactement en p *ItemTests* où exactement x choix possibles sont associés à chacun et dont exactement un (qui est le minimum requis), le nombre des sujets d'évaluation différenciés représenté dans l'équation ci-dessus est ensuite simplifié en :

$$ {}_pP_p \,.\, 1 \,.\, {}_{x-1}P_{x-1} \;\equiv\; p! \,.\, (x-1)! \tag{3.4} $$

Combinatoirement, ceci procure déjà un grand éventail de disparité possible, mais on ne connaît pas pour autant la distance moyenne procurer, donc on ne peut pas assurer un minimum de différenciation à l'enseignant. Dans la sous-section suivante, on présente le protocole d'expérimentation de la méthode de génération suivant *RandomGenerateTest()*.

3.3.2 Protocole d'expérimentation

L'objectif de cette expérimentation est de mesurer la performance de cet algorithme au regard de la différenciation. Pour cela, on veut pouvoir générer des *Patrons de Collections* de *Tests* différenciés. On entend par "*Patron de Collection*", une *Collection* de *Tests* qui est fait que d'identifiants de questions et de choix de réponses. Ainsi, le *Patron* est généré une fois, mais permet de créer différentes *Collections* de *Tests* par la suite, car seulement les contenus des questions et choix de réponses sont rajoutés, peu importe l'enseignement. L'avantage est qu'un enseignant peut utiliser un même "*Patron de Collection*" pour générer des *Collections de Tests* d'évaluation pour des matières diverses, dans un temps réduit. Cette notion de "*Patron de Collection*" est développée d'avantage dans le prochain chapitre.

Un *Patron de Collection* regroupe un ensemble de *Tests* et chaque *Test* est représenté par des identifiants d'*ItemTests* (énoncés) et de *RepItemITs* (choix de réponse). La figure 3.12 présente l'exemple d'un *patron de Test* qui est fait de 3 *ItemTests* et de 4 *repItemITs* par *ItemTest*. Au début de chaque chaîne sont représentés les identifiants des énoncés (encerclés en vert) et les identifiants de choix de réponses (encerclés en rouge). Ainsi, un *Patron de Collection* qui est fait de 5 *Tests* de 3 questions contiendra 5 *Tests* plus ou moins similaires à 3.12.

Ainsi, le point primordial est de trouver une configuration dans laquelle la différenciation soit maximisée et la taille de la base source soit minimisée le plus possible. Dans la suite du manuscrit, nous parlerons parfois de *Collection* tout simplement, pour faire allusion à un *Patron de Collection*.

Cette expérimentation est faite en deux temps, avec deux protocoles différents. La première partie met l'emphase sur les performances de l'algorithme aléatoire en terme de différenciation quand les énoncés *ItemTests* varient. C'est-à-dire, la différenciation possible quand la quantité d'énoncés varie dans la base et dans les *Tests*. La deuxième partie de l'expérimentation considère principalement les *RepItemITs* pour mesurer l'impact que peuvent avoir les choix de réponses sur la différenciation. Les protocoles d'expérimentation s'appuient sur les quatre paramètres suivants :

- $X1$ représente la taille de la base d'*ItemTestPatterns*. Cette taille correspond à la quantité d'*ItemTestPatterns* dont dispose l'enseignant dans sa base de données.

FIGURE 3.12 – Exemple d'un *patron de Test* de 3 questions

- $X2$ représente la taille du *Patron de Collection* de *Tests* à générer. Cette taille correspond à la quantité de *Tests* que l'enseignant souhaite générer.
- $X3$ représente la taille de chaque *Test*. Cette taille correspond en la quantité d'*ItemTests* que l'enseignant souhaite avoir dans chaque *Test*.
- $X4$ représente la quantité de *RepItemIts* par *ItemTest*. Donc, la quantité de choix pour chaque *ItemTest*.

Pour la première partie d'expérimentation, nous avons paramétré $X4$ à 4, c'est à dire, que l'on considère que tous les *ItemTests* ont 4 choix de réponses, et nous supposons que tous les *ItemTestPatterns* ont 6 choix de réponses. Ainsi, nous avons généré des *Patrons de Collections* de *Tests*, en faisant varier $X1$, $X2$ et $X3$ sur des ensembles discrets de valeurs choisies empiriquement : $X1$ varie sur l'intervalle [10,250]; $X2$ varie sur l'intervalle [10,250] et $X3$ sur l'intervalle [2,300]. Nous obtenons ainsi un *Patron de Collection* pour chacun des triplets *(X1, X2, X3)*, *Patron de Collection* dans lequel nous calculons la distance pour tous les couples de *Tests*. A partir des distances obtenues pour chaque *Patron de Collection*, la distance moyenne, minimale et maximale sont déterminées. Par exemple, la figure 3.13 montre les courbes pour un *Patron de Collection* de 100 *Tests*, formés de dix questions, provenant d'une base de 10 questions source. En observant les légendes, on remarque que la différenciation moyenne (courbe bleue) dans ce *Patron de Collection* se stabilise autour de 0.6.

Ainsi, 1694 *Patrons de Collections* ont été générés. Les distances moyennes, minimales, maximales et écartypes de tous les *Patrons* qui sont issus d'une même taille de base

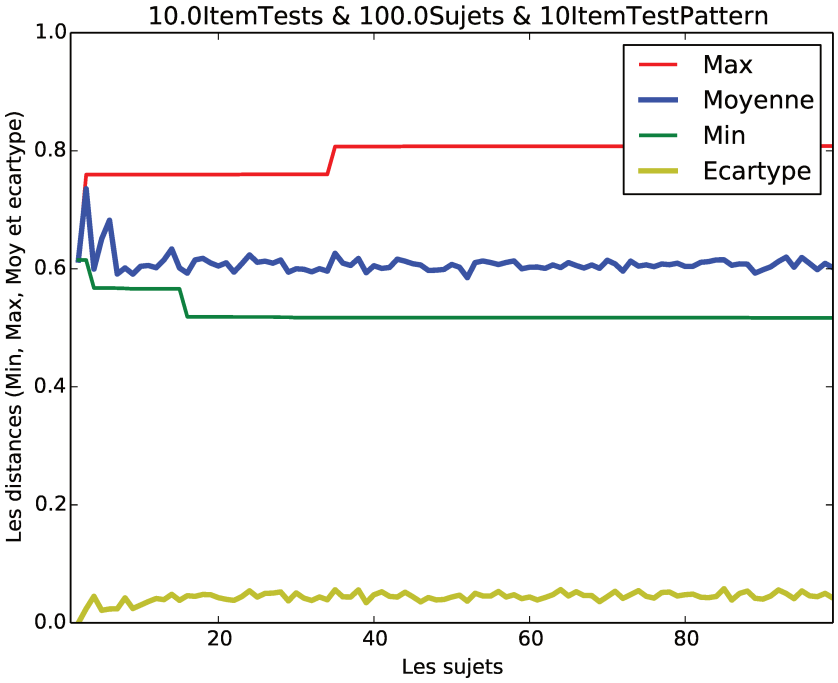

FIGURE 3.13 – Exemple des distances *DTest()* pour un *Patron de Collection* de 100 *Tests*, issu d'une base de 10 *ItemTestPatterns* et formés de 10 *ItemTests*

d'*ItemTestPatterns* et qui contiennent une même quantité d'*ItemTests* dans un *Test* ont été regroupés et représentés sur une même courbe. Par exemple, la figure 3.14 représente le regroupement de tous les *Patrons de Collections* générés à partir d'une base de 10 *ItemTestPatterns* et contenant 10 *ItemTests*. Les étoiles représentent les distances moyennes, les ronds sont les distances maximales, les carrés sont les distances minimales et les losanges sont les écartypes. Par exemple, les *Patrons de Collections* de taille 200 ont une distance maximale de 0.6, une distance moyenne de 0.44 et une distance minimale de 0.37.

Ces regroupements ont donné lieu à 121 courbes comme celle de 3.14 où les distances moyennes, minimales et maximales sont représentées (avec une échelle de 0 à 1 en ordonnée et la quantité de *Tests* par *Collection* (de 2 à 300) en abscisse). En vue de voir si le rapport entre la taille d'une base de questions sources et la quantité de questions dans les sujets générés (*(ItemTestPatterns/ItemTests)*) a un impact sur la différenciation moyenne dans les *Patrons de Collections*, les 121 courbes sont subdivisées en des sous-catégories de courbes représentant un même rapport. Pour la suite, nous parlerons de "ratio" 3.5 au lieu de "rapport". Cette subdivision a permis de classer chaque sous-catégorie par rapport à son ratio sur une même courbe. Par exemple, la figure 3.15 illustre la représentation

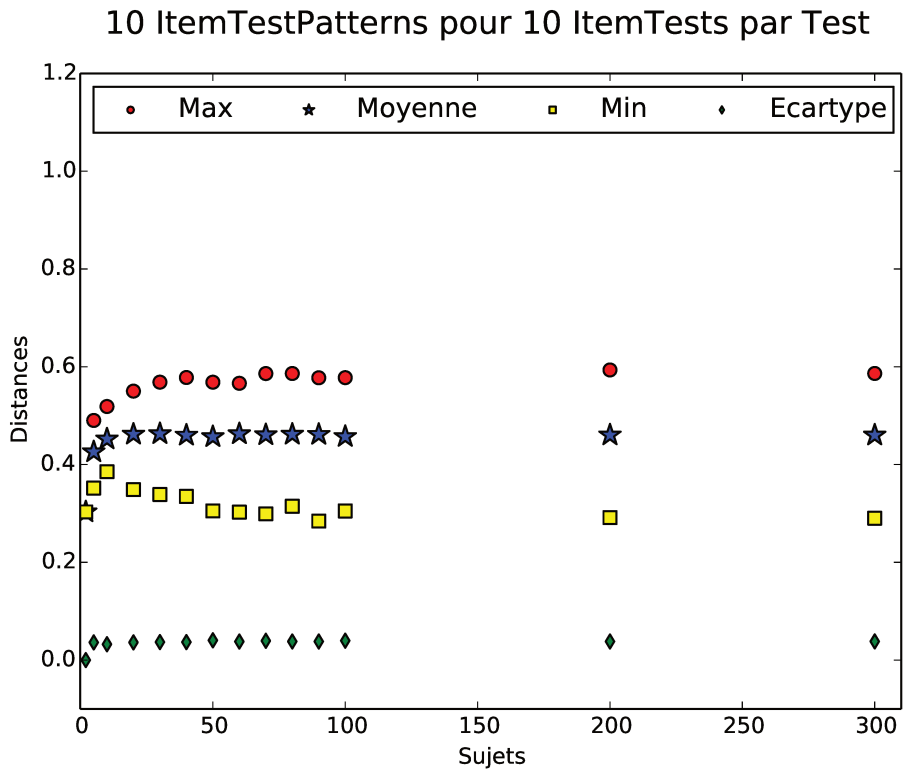

FIGURE 3.14 – Exemple du regroupement des *DTest()* pour les *Patrons de Collections* issus d'une base de 10 *ItemTestPatterns* pour des *Tests* formés de 10 *ItemTests*

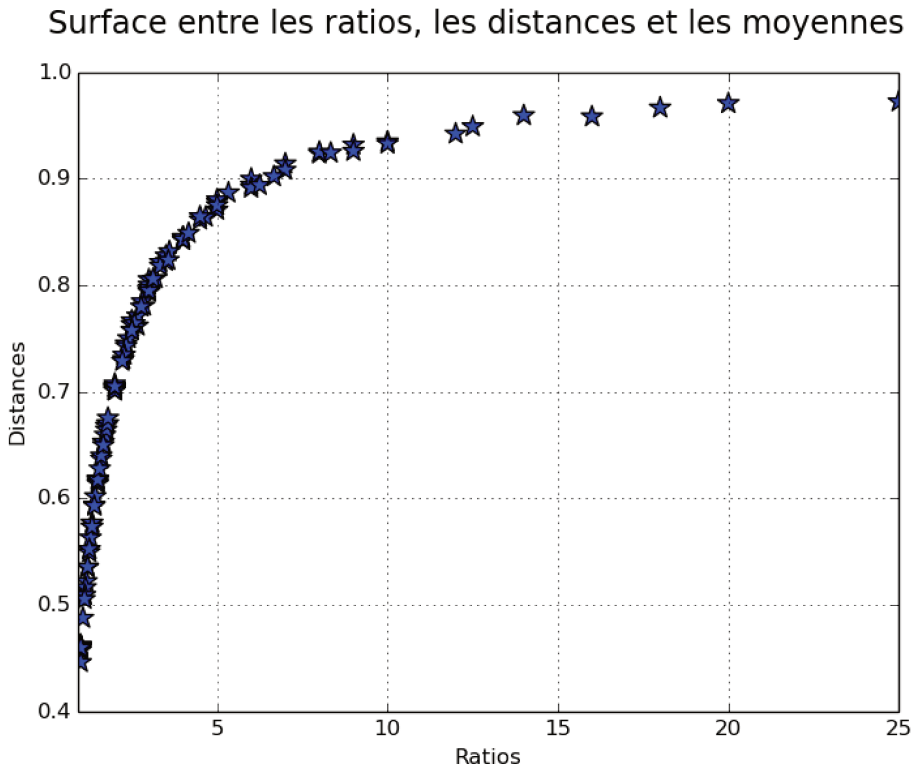

FIGURE 3.15 – Nuage de points des distances moyennes dans les Collections regroupées par ratio allant de 1 à 25

des distances moyennes des *Patrons de Collections* allant de 0 à 1 par rapport aux ratios allant de 1 à 25. On peut observer que pour un ratio de 10, la différenciation moyenne dans un *Patron de Collection* est supérieure à 0.9 (ce qui signifie que le fait d'avoir dix fois plus de contenus dans sa base que sur les *Tests* générés peut procurer une très grande différenciation moyenne en utilisant un générateur aléatoire).

Comme nous l'avions spécifié dans le début de la section, la deuxième partie d'expérimentation a pour objectif de voir si la quantité de choix de réponses peut influer la différenciation aussi. Pour vérifier cela, nous avons fait de nouvelles expérimentations en suivant le protocole suivant : les *Patrons de Collections* générés contiennent 100 *Tests* ; chaque *Test* est fait de 10 *ItemTests* ; la base source contient 10 *ItemTestPatterns*. C'est-à-dire, on a gardé une même taille de base source, un même quantité de questions dans les *Tests* et une même taille de *Patron de Collection*, puisque qu'on ne souhaite faire varier que les jeux de réponses. Ainsi, les *RepItemITPs* (choix de réponses dans la base) varient sur l'intervalle de [3,20] et les *RepItemITs* (choix de réponses dans les *Tests*) sur [2,10]. Après l'obtention des distances pour chaque *Patron de Collection*, les mêmes démarches et calculs sont faites que ceux de la première expérimentation.

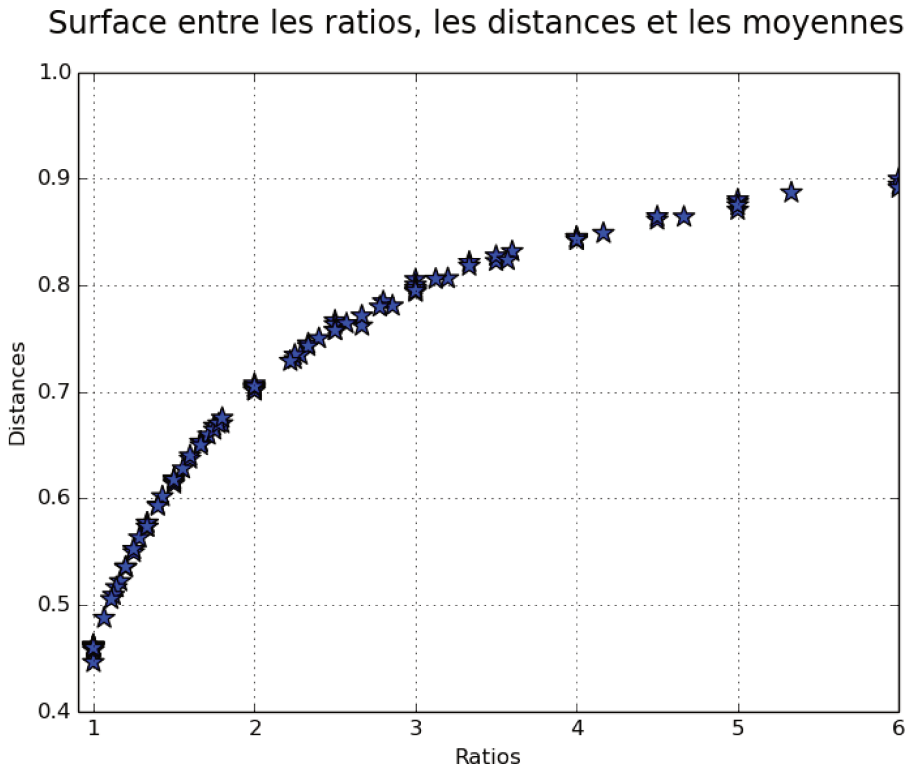

FIGURE 3.16 – Distances moyenne comparées au ratio

3.3.3 Résultats pour les deux expérimentations

La première observation des résultats nous a permis de comprendre que la distance moyenne entre les *Tests* d'une même *Collection* est proportionnelle au ratio entre *ItemTestPatterns* dans la base de données source et *ItemTests* dans les tests :

$$Ratio = \frac{\#ItemTestPattern \in Source\ Database}{\#ItemTest \in Test_i}. \tag{3.5}$$

Par ailleurs, en analysant les courbes on constate que pour avoir une grande différenciation dans sa *Collection* de *Tests* différenciés avec l'algorithme aléatoire, l'enseignant doit disposer d'une grande taille de base de questions sources. En d'autres termes, si pour un ratio de 10 nous avons une différenciation moyenne de 0.9, pour un ratio de 25 on a une différenciation très proche de 1 et pour un ratio de 5 on a une différenciation moyenne de supérieure à 0.8 mais inférieure à 0.9. Cette remarque est illustrée dans la figure 3.15 et aussi la figure 3.16 qui fait un zoom sur les différenciations moyennes pour les ratios allant de 1 à 6.

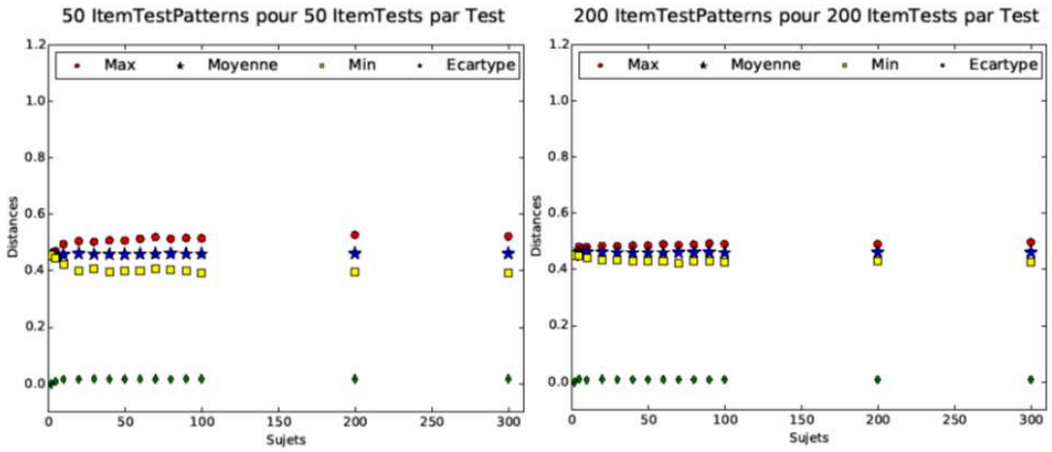

FIGURE 3.17 – Comparatif des collection de *Tests* de tailles différentes pour un même ratio

Le nuage de points 3.16 nous a permis d'observer également que lorsque le ratio est supérieur à 3, la génération aléatoire fournit une distance moyenne supérieure à 0,8 entre les couples de *Tests* générés, ce qui signifie qu'il existe un taux de disparité élevé dans les *ItemTests* des *Tests*. Ainsi, l'enseignant qui veut avoir une différenciation moyenne supérieure ou égale à 0,8 en utilisant un générateur aléatoire, doit avoir au minimum trois fois plus de questions dans sa base que dans ses *Tests*. Par exemple, il lui faudrait 180 questions sources pour avoir une différenciation moyenne proche de 0.8 si chaque *Tests* contient 60 questions.

Nous remarquons également pour un même ratio, même si les bases de données de *ItemTestPattern* ont des tailles différentes, la distance moyenne est la même. Par exemple, générer 100 *Tests* avec 10 *ItemTests* choisis parmi un ensemble de 20 *ItemTestPatterns* fournit la même distance moyenne de 0.78 que de générer 100 *Tests* avec 30 *ItemTests* choisis dans un ensemble de 60 *ItemTestPatterns*. La figure 3.17 présente les distances moyennes obtenues avec la génération aléatoire, par rapport à divers ratios.

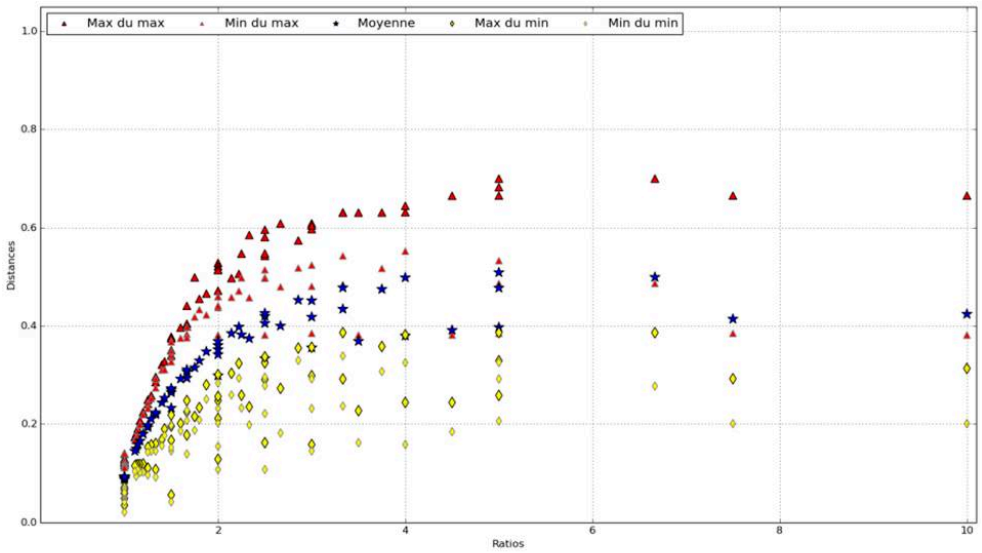

FIGURE 3.18 – Augmentation de la distance moyenne grâce à l'augmentation des choix de réponses

Concernant l'expérimentation qui a été menée sur les choix de réponse, on a observé qu'une augmentation des choix dans les *ItemTestPatterns* peut causer une augmentation de la différenciation dans une *Collection*. Par exemple, si on met 10 *RepItemITPs* dans chaque *ItemTestPattern* au lieu de 6, on peut observer une augmentation de la distance maximale de 0.4 à 0.7 comme illustré dans la figure 3.18. En d'autres mots, l'enseignant qui ne dispose pas d'une quantité de questions sources importante, peut augmenter la différenciation dans sa *Collection* en augmentant le nombre de choix possibles par question source. Ainsi, pour obtenir une bonne différenciation moyenne avec un générateur de type aléatoire sans pour autant avoir beaucoup de questions dans sa base, il peut être judicieux d'augmenter son de choix de réponses.

3.4 Synthèse

Ce chapitre a permis d'exposer la métrique *DTest()* qui a été conçue pour mesurer la différenciation entre deux *Tests* au sein d'une même *Collection*. Elle permet de prendre en compte les différents éléments et sous éléments qui constituent un *Test* à savoir les questions et les choix. Nous proposons deux fonctions grâce auxquelles la disparité et la permutation sont calculées entre les questions et les réponses, ce qui permet d'avoir une

valeur finale entre 0 et 1 représentant le niveau de différenciation entre les Tests comparés. Une première démarche expérimentale illustre le comportement attendu de *DTest()*.

Le deuxième objectif de ce chapitre a été de présenter un exemple d'implantation de *DTest()* dans un algorithme de référence de génération pour d'une part vérifier que *DTest()* répond correctement aux distances imaginées pédagogiquement selon le degré de permutation et de disparité entre les deux *Tests* comparés ; d'autre part, nous avons pu mesurer les performances en terme de différenciation de la méthode de génération par l'algorithme aléatoire. Cela nous a révélé que l'algorithme aléatoire permet d'obtenir une différenciation moyennement grande (supérieure à 0.8) quand l'enseignant dispose de trois fois plus de contenus dans sa base source que dans ses *Tests*. Donc, ses limites en terme de différenciation commence à partir des ratios inférieurs à 3.

Dans le but de maximiser la différenciation possible dans un *Patron de Collection* de *Test* en minimisant la taille de la base source, le chapitre suivant présente deux nouvelles approches algorithmiques développées dans le cadre de cette thèse. Par rapport aux résultats présentés sur l'algorithme aléatoire, la question qu'on répond dans le prochain chapitre est : est-il possible de maintenir de bons niveaux de différenciation moyenne pour des ratios inférieurs à 3 ?

Chapitre 4

Génération de sujets différenciés

Ce chapitre a pour objectif de présenter deux nouvelles méthodes/approches de génération de *Tests* différenciés. Elles sont développées autour de la métrique *DTest()*, ce qui permet de contrôler la différenciation entre les *Tests* au moment de leur génération. Le principal but de ces méthodes est de maintenir de bons niveaux de différenciation dans les *Patrons de Collections* de *Tests* générés, depuis une base de questions restreinte. Plus spécifiquement, ces deux méthodes ont pour mission d'augmenter la distance moyenne dans les ratios ($ItemTestPattern/ItemTest$) inférieurs à trois (3), car, comme nous l'avons vu dans le chapitre précédent, le tirage aléatoire offre de bonnes garanties de distance structurelle pour les ratios supérieurs à 3.

Une première section est consacrée à la présentation d'une méthode de génération basée sur un algorithme par Sélection. Une deuxième section présente une méthode de génération basée sur un algorithme qui est construit sur une méta-heuristique. Il s'agit d'une adaptation de notre problème en un algorithme génétique. Pour ces deux algorithmes, leur fonctionnement, leur expérimentation et les résultats sont présentés. Après une analyse et une comparaison des résultats des trois différentes méthodes de génération, une présentation du prototype envisagé est faite en fin de chapitre.

4.1 Génération par la sélection

L'objectif de cette méthode est d'explorer si la différenciation moyenne peut être meilleure sur les *Patrons de Collections* générés dans les ratios inférieurs à 3. Elle est construite sur un algorithme qui existe en deux variantes que nous dénommons *Sélection HARD* et *Sélection SOFT*. D'un point de vue global, pour la constitution de chaque *Test*, l'algorithme par Sélection suit le principe de la génération aléatoire, à savoir, le tirage aléatoire des énoncés et des choix de réponses. Toutefois, la distance de chaque *Test* généré est calculée avec tous les *Tests* préalablement pour vérifier s'il est suffisamment différents

des autres. Ainsi, le but est de n'avoir aucun couple de *Tests* ayant une distance en dessous d'un certain seuil prédéfini à l'issue de la génération du *Patron de Collection*.

4.1.1 Algorithmes par Sélection : HARD et SOFT

L'algorithme par *Sélection Hard* est pour une sélection stricte. On entend par stricte, le fait de permettre à l'enseignant de prédéfinir une distance seuil unique de manière empirique dans l'intervalle [0, 1]. Ainsi, la distance de chaque *Test* généré est calculée avec tous les *Tests* générés avant lui et est comparée au seuil. Comme il est présenté à travers l'algorithme 3, si au moins une de ces distances est inférieure ou égale au seuil défini, le *Test* généré est catégoriquement rejeté et la constitution d'un nouveau commence. Chaque *Test* est accepté et ajouté dans le *patron de Collection* si et seulement si sa distance avec tous les autres est supérieure ou égale au seuil. Ce processus est répété autant de fois qu'il y a de *Test* à générer.

Algorithm 3 Génération par la selection HARD

Require:
  $threshold$: {seuil prédéfini}
  db : {base source contenant les ItemTestPatterns}
  $nbOfItemTests$: {nombre d'ItemTests par Test}
  $nbOfTests$: {nombre de Tests par collection}
  c : {un collection}
  $test$: {un Test}
Ensure:
  while $\#c \leq nbOfTests$ **do**
    $test \leftarrow randomGenerateTest(db, nbOfItemTests)$
    for all $t \in c$ **do**
      if $DTest(test, t) \geq threshold$ **then**
        $c \leftarrow c \cup \{test\}$
      end if
    end for
  end while
  return c

L'algorithme 3 précise que la méthode HARD doit disposer d'un ensemble de paramètres tels que : le seuil prédéfini ; la base source d'*ItemTestPatterns* ; la taille des *Tests* ; et la quantité de choix par *ItemTest*. Ensuite, tant que tous les *Tests* de la *Collection* ne sont pas générés, l'algorithme continue son processus de génération aléatoire ; de vérification des distances ; puis d'acceptation ou de rejet. Si le temps d'exécution est atteint et que la génération est stoppée alors que la taille du *patron de Collection* n'était pas encore complète, tous les *Tests* déjà générés sont détruits.

L'algorithme par Sélection SOFT (cf. 4) reprend le même processus que l'approche HARD en permettant à l'enseignant de définir un intervalle de tolérance en dessous du seuil unique. Ceci permet de relaxer la condition d'acceptation d'un *Test* pour favoriser l'aboutissement de la génération de la *Collection*. Cet intervalle est obligatoirement définie entre zéro et un [0,1]. Comme pour l'approche HARD, à la génération de chaque *Test*, sa distance est calculée avec les *Tests* générés avant lui. Toutefois, pour que le *Test* soit accepté et ajouté dans la *Collection*, sa distance avec tous les autres *Tests* doit être soit incluse dans l'intervalle.

Algorithm 4 Generation by selection SOFT

Require:
  $thresholdmin$: {seuil minimal prédéfini}
  $thresholdmax$: {seuil maximal prédéfini}
  $nbOfItemTests$: {nombre d'ItemTests par Test}
  $nbOfTests$: {nombre de Tests par collection}
  c : {un collection}
  $test$: {un Test}
Ensure:
  while $\#c \leq nbOfTests$ **do**
    $test \leftarrow randomGenerateTest(db, nbOfItemTests)$
    for all $t \in c$ **do**
      if $DTest(test, t) \in [thresholdmin, threshold - max]$ **then**
        $c \leftarrow c \cup \{test\}$
      end if
    end for
  end while
  return c

Cette méthode permet d'être moins exigeant sur le seuil de différenciation, et facilite l'acceptation des nouveaux *Tests* générés. La version SOFT de la sélection a été conçue et développée parce que la version HARD est rigide et peut ne pas aboutir, ce qui suppose une annulation complète de tout le *Patron de Collection*. La sous-section suivante présente le protocole d'expérimentation de l'algorithme de Sélection et les résultats obtenus.

4.1.2 Expérimentations et résultats

Pour évaluer cet algorithme, nous avons choisi de prendre en compte les ratios inférieurs à quatre. Aussi, sachant que tous les *Patrons de Collections* issus d'un même ratio ont une distance moyenne sensiblement égale, nous avons réduit la quantité des différentes tailles des *Patrons de Collections* à générer pour cette expérimentation. Au final, les paramètres de génération sont fixés ainsi :

- $X1$ représente la taille de la base d'*ItemTestPatterns*. Cette taille correspond en la quantité d'*ItemTestPatterns* dont dispose l'enseignant dans sa base de données. Ce paramètre a été fixé à 50.
- $X2$ représente la taille de *Patron de Collection* de *Tests* à générer ; Cette taille correspond à la quantité de *Tests* que l'enseignant souhaite générer. Ce paramètre peut prendre une de ces trois valeurs *[40, 150, 1000]*.
- $X3$ représente la taille de chaque *Test*. Cette taille correspond à la quantité d'*ItemTests* que l'enseignant souhaite avoir dans chaque *Test*. Ce paramètre varie de *10 à 50*
- $X4$ représente la quantité de *RepItemIts* par *ItemTest*, donc, la quantité de choix pour chaque *ItemTest*, le paramètre est fixé à 4.

Ainsi, lors des expérimentations de l'aléatoire, pour chaque couple de *Tests* de chaque *Patron de Collection*, la distance moyenne est calculée, ainsi que la distance maximale et la distance minimale. Les figures 4.1 et 4.2 représentent les distances moyennes (symboles étoiles), maximales (symboles triangles rouges) et minimale (symboles losanges jaunes) pour les méthodes HARD et SOFT. Nous remarquons que la différence entre les distances moyennes des algorithmes HARD et SOFT est de l'ordre du millième, donc pas de différence significatives en termes de performance. Toutefois, la méthode SOFT prend un avantage sur la méthode HARD dans le fait que ses générations aboutissent plus facilement, contrairement à celles de HARD qui sont conditionnées par un seuil unique. Donc, la Sélection SOFT procure un gain de temps d'exécution, elle assure un aboutissement et propose une distance moyenne légèrement plus faible que celle de HARD. Pour la méthode HARD, on a également remarqué que le seuil est très sensible, pour expliquer cela avec un exemple, une augmentation du seuil de environ 0,001 peut être responsable d'une augmentation du temps de génération de 1 heure à 24 heures.

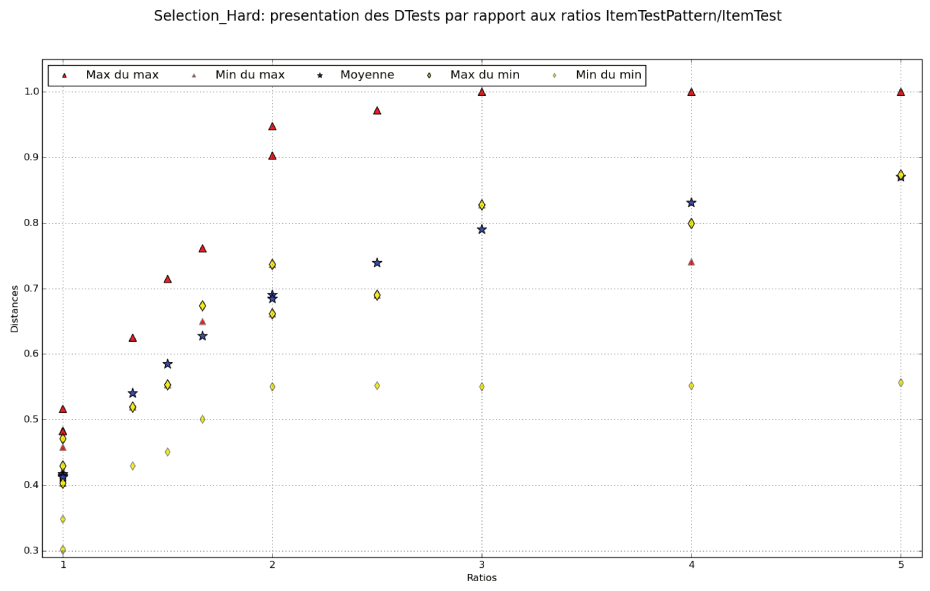

FIGURE 4.1 – Les distances avec la Sélection HARD

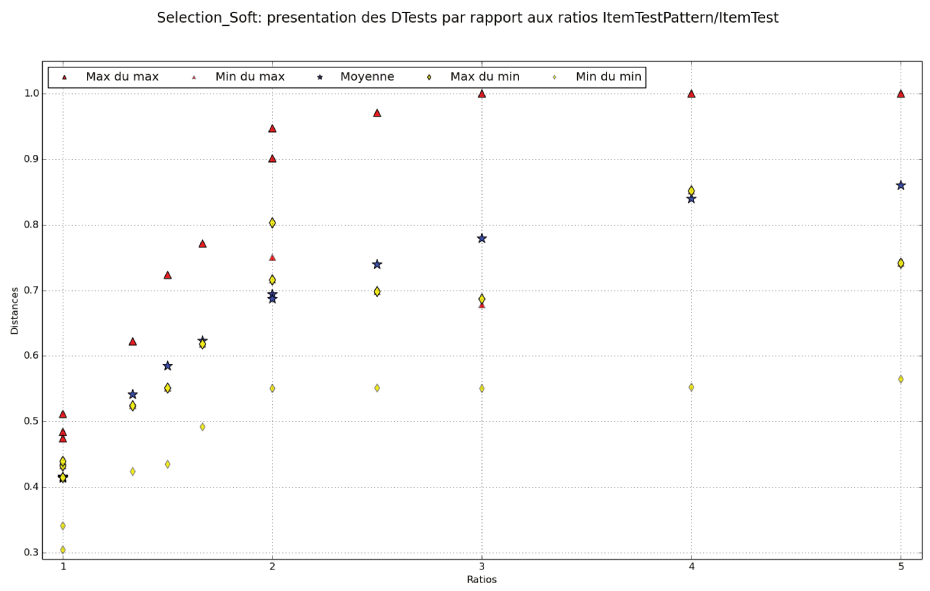

FIGURE 4.2 – Les distances avec la Sélection SOFT

La figure 4.3 présente un comparaison entre les méthodes de génération Aléatoire et Sélection. Elle montre que la distance moyenne avec la Sélection est supérieure à la celle issue de l'aléatoire. Donc, sur un point de vue global, la distance moyenne obtenue avec la génération par sélection est environ 5% supérieure à la distance moyenne résultant de la

génération aléatoire.

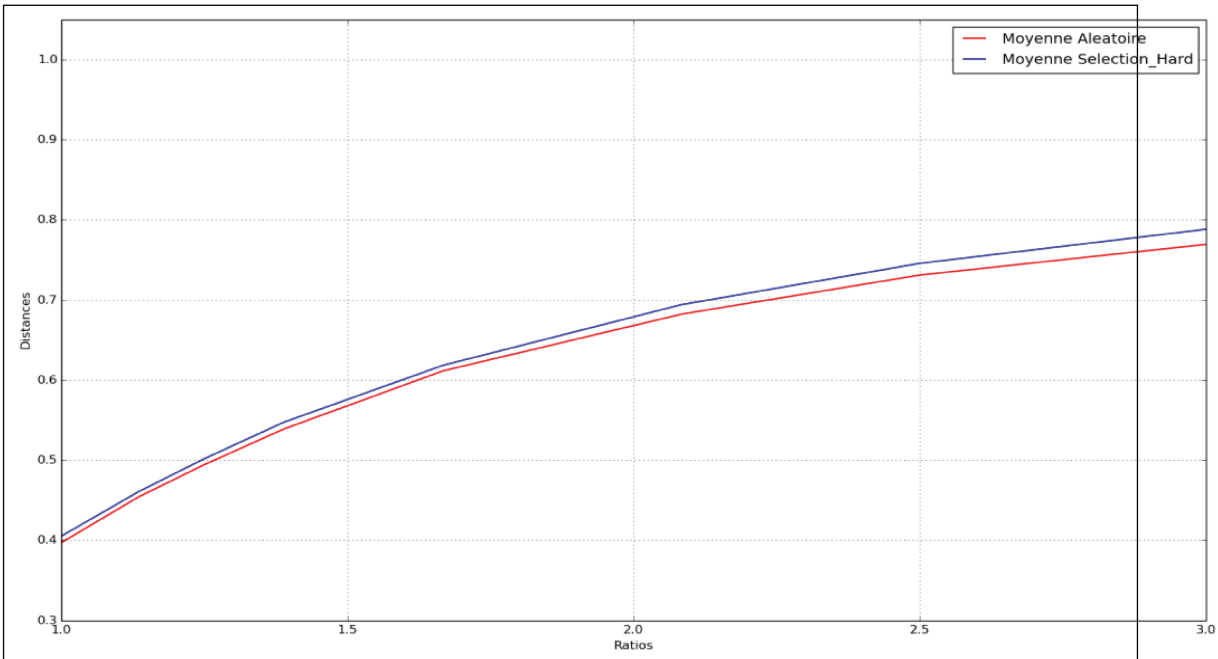

FIGURE 4.3 – Les distances moyennes de la sélection face à l'aléatoire (ratio 1-3)

Le principal avantage de la méthode de génération par la Sélection par rapport à celle de l'Aléatoire s'observe plus localement. La Sélection permet d'avoir un ensemble de *Tests* dont la distance minimale qu'on peut avoir entre deux *Tests* d'un même *Patron de Collection* ne peut être inférieure à un seuil prédéfini, ce qui est différent dans le cadre de la génération aléatoire. Par exemple, la figure 4.4 pourrait être analysée ainsi : si on doit générer aléatoirement un *Patron de Collection* de 40 *Tests* de 50 *ItemTests* chacun, issus d'une base de 50 *ItemTestPatterns* (ratio 1), en vérifiant la *DTest()* entre tous les 40 *Tests* deux à deux (780 couples), il est possible d'avoir des couples de *Tests* qui soient distants de 0.44, tout comme on peut avoir des couples de *Tests* qui soient distants de 0.37. Par contre, si on doit générer cette même *Collection* de *Tests* par la méthode de la Sélection pour un seuil de 0.407, la *DTest()* minimale qu'on aura pour tout couple de cette *Collection* sera de 0.407. En superposant ces deux *Collections*, on trouve que 27% des *Tests* de la *Collection* qui a été générée de façon aléatoire serait rejeté si on devait filtrer cette *Collection*. C'est-à-dire, dans la *Collection* générée avec l'algorithme aléatoire, 27% des distance entre les couples de *tests* est inférieur à 0.45. Donc, l'écartype de l'aléatoire est supérieur à celui de la sélection. Ce cas de figure est également illustré à travers la figure 4.5 qui représente une *Collection* de 30 *Tests* générée à partir d'un ratio de 1.25. De cela on comprend que la distance moyenne dans un *Patron de Collection* de *Tests* générée avec l'algorithme de Sélection est plus important que l'algorithme aléatoire.

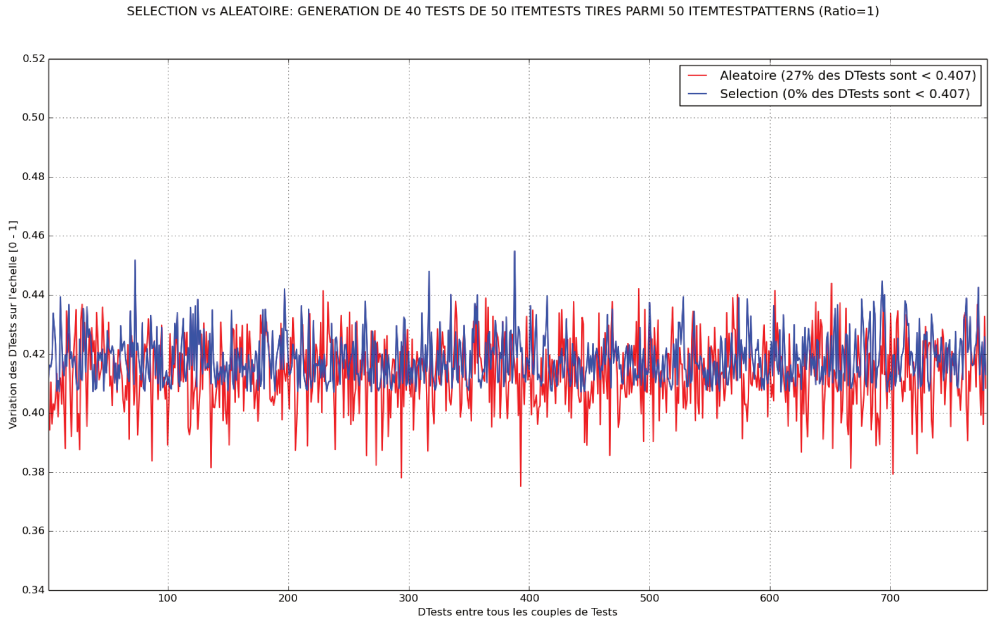

FIGURE 4.4 – Comparaison des distances entre les couples de *Tests* issues d'une collection de 40 *Tests*, faite l'algorithme Aléatoire et celui de la Sélection

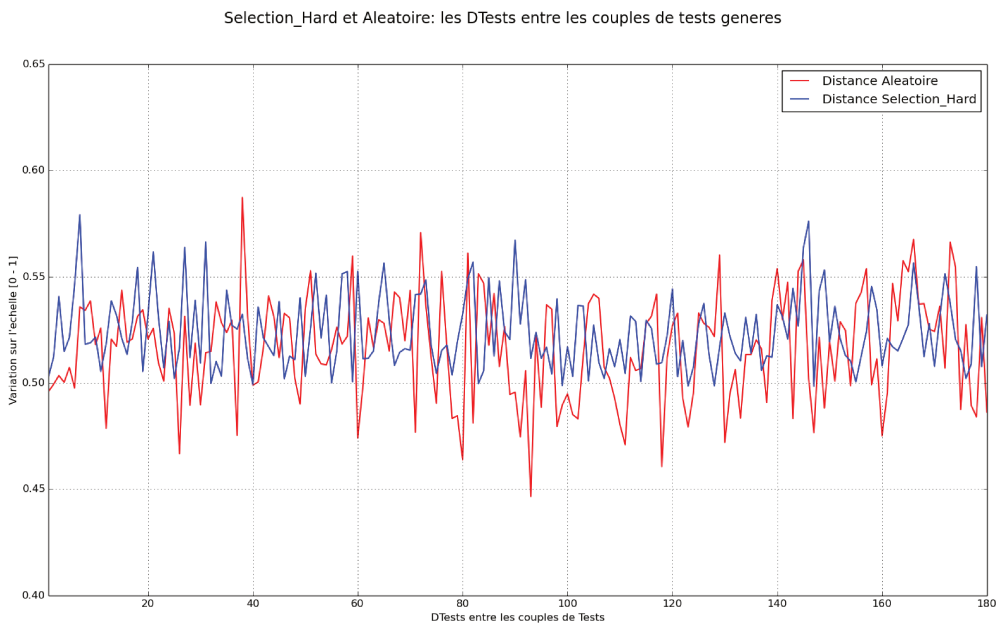

FIGURE 4.5 – Comparaison des distances entre les couples de *Tests* issues d'une collection de 30 *Tests*, faite l'algorithme Aléatoire et celui de la Sélection

Se basant sur ces expérimentations, le tableau 4.6 présente une recommandation sur la façon de fixer les seuils lors de l'utilisation de la méthode de Sélection pour différents cas de figure. Ces cas de figures nous montrent que la pré-définition des seuils est liée principalement aux ratios et la quantité de *Tests* qu'on souhaite générer. Ces intervalles sont déterminés grâce aux différentes expérimentations que nous avons faites avec cet algorithme. Par exemple, ce tableau recommande d'utiliser un seuil entre 0.4 et 0.405 pour un ratio de 1, ce qui garantit l'aboutissement de la génération de la *Collection* de *Tests*. Ce tableau prend en compte que des expérimentations qui ont été menées sur des *Collections* de tailles variant de 30 à 300 *Tests*.

Ratios et Seuils	
Collections de 30 à 300 Tests	
Ratios	Seuil Sélection
1	[0.400 – 0.405]
1.2	[0.400 – 0.500]
1.5	[0.400 – 0.555]
1.7	[0.400 – 0.590]
2	[0.400 – 0.650]
3	[0.400 – 0.730]

FIGURE 4.6 – Table représentant les seuils à partir desquels la sélection aboutit toujours

En résumé, la sélection dans ces deux variantes permettent d'optimiser la différenciation quand le ratio est faible. Toutefois, le comparant avec l'aléatoire, la différenciation moyenne est améliorée de 5%. De ce fait, on se demande s'il est possible de faire mieux avec une nouvelle méthode de génération. C'est dans cette optique qu'on a proposé l'approche méta-heuristique à travers un algorithme génétique qui est présenté dans la prochaine section.

4.2 Génération construite sur une méta-heuristique

Disposant de la métrique *DTest()*, nous nous appuyons sur la piste de résolution de la contrainte de différenciation par une méta-heuristique, plus particulièrement un algorithme génétique. Une méta-heuristique est un type d'algorithme qui permet de résoudre des problèmes d'optimisation difficile [17]. Il nous permet d'interroger le fait de garantir

de meilleures différenciations dans les *patrons de Collections* générés comparativement aux méthodes de génération Aléatoire et Sélection. Les sous-sections suivantes présentent une méthode de génération s'appuyant sur un algorithme génétique.

4.2.1 Principe de l'algorithme génétique

Un algorithme génétique est une méta-heuristique utilisée pour obtenir des solutions approchées lorsque le temps nécessaire à l'exhibition d'une solution optimale est au delà du raisonnable (i.e. complexité polynomiale) [64]. Par analogie avec la biologie, cet algorithme permet de partir d'une première population d'individu quelconque qu'il améliore après plusieurs itérations en considérant une fonction d'évaluation (fonction de Fitness) et en appliquant des opérations spécifiques, jusqu'à obtenir une population proche de celle attendue. Pour faire évoluer les populations de solution de manière progressive, les principales opérations mises en place dans un algorithme génétique à chaque itération sont : premièrement, la sélection/tri qui consiste en un classement des individus de la population en raison de certains critères prédéfinis. Cela s'approche du concept de la sélection naturelle qui stipule que les individus les plus adaptés vivent et se croisent, tandis que les moins adaptés meurent avant la reproduction [26]. Cette opération est possible grâce à une fonction de Fitness qui joue un rôle d'évaluateur des individus de la population [70] ; deuxièmement, l'opération de suppression qui consiste en la destruction/mort d'un pourcentage des individus les moins adaptés. Ce pourcentage est défini en amont par le concepteur ; troisièmement l'opération de croisement qui s'occupe de compléter la population avec de nouveaux individus après l'opération de suppression. Généralement, les individus les plus adaptés échangent des parties de leurs chaînes pour donner naissance à de nouveaux individus [72] ; et quatrièmement l'opération de mutation qui permet de substituer de façon aléatoire un gène à un autre dans un individu. Cette dernière se fait généralement à un taux très faible pour éviter de tomber dans une recherche trop aléatoire, ce qui nuirait à l'évolution de la population []. Les algorithmes génétiques font partie d'une famille d'algorithmes évolutionnistes car leur but est d'obtenir une solution approchée à un problème d'optimisation en faisant évoluer sa population par rapport à un certain critère d'évaluation défini au préalable.

Dans notre cas, on cherche à générer des populations de *patrons de Collections* de *Tests* et les mieux adaptés sont les plus distants, c'est-à-dire qui ont une meilleure différenciation moyenne. Comme dans l'algorithme aléatoire, on définit les critères de génération en spécifiant tous les paramètres nécessaires comme : un nombre d'itérations ; un temps d'exécution maximal (timeout) ; la taille de la population (où les individus sont des *patrons de Collections*) ; les pourcentages de suppression et de mutation à chaque itération. Pour son fonctionnement, une première population de *patrons de Collections* de *n Tests* est générée aléatoirement et *DTest()* la moyenne de chaque *Patron de Collection* est calculée. La

population est triée et ordonnée du *Patron de Collection* le plus faible au plus fort en raison de leur distance moyenne. Après le tri des individus de la population, la suppression d'un pourcentage d'individus faiblement distants est effectuée. Comme l'illustre la figure 4.7, nous considérons une population contenant n individus et le pourcentage de suppression est fixé empiriquement à 8%. Ensuite, des croisements sont effectués entre des individus plus fortement distants en échangeant la moitié de leurs constituants (4.8) pour créer un nouvel individu. Les croisements sont effectués jusqu'à ce que la population redevienne complète.

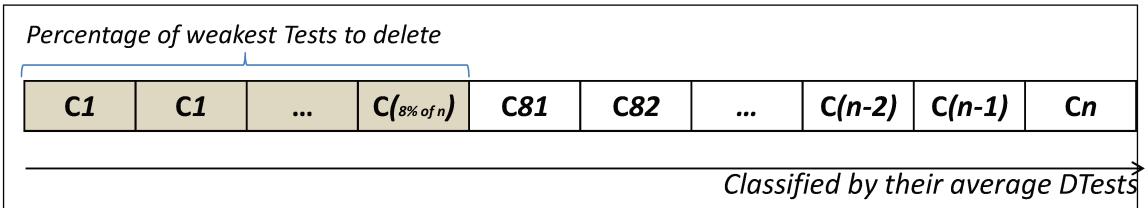

Figure 4.7 – Principe de l'opérateur de suppression (8%)

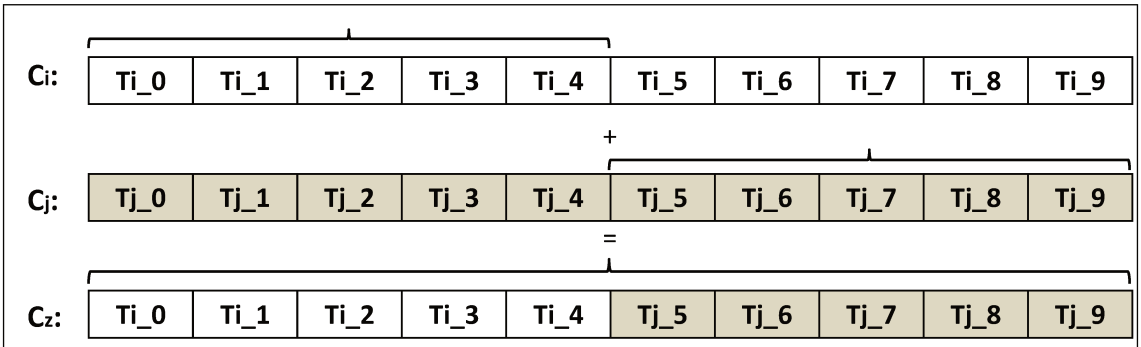

Figure 4.8 – Principe de l'opérateur du crossover

À la fin du processus (génération des individus, tri des individus, suppression des individus faibles et croisement entre les individus forts), une mutation est effectuée une fois sur trois. Pour ce faire, nous supprimons aléatoirement le pourcentage prédéfini de mutation de la population et nous générons aléatoirement ce même pourcentage pour compléter la population. L'ensemble du processus est répété jusqu'à ce que le délai imparti ou la quantité d'itérations prédéfinie soit atteinte. L'algorithme 5 l'illustre et montre que la solution attendue dans notre cas est une *Collection* de *Tests* avec une grande différenciation entre chaque paire de *Tests*. Les *Collections* représentent les individus de notre population et la métrique *DTest()* est utilisée comme fonction de fitness et appliquée à chaque *Collection* afin d'évaluer sa performance en termes de différenciation. Cependant, le temps d'exécution peut aller jusqu'à 2 heures de plus que l'algorithme aléatoire lorsque le ratio est inférieur à 3 et que le seuil de différenciation attendu est assez élevé.

Algorithm 5 Algorithme génétique : geneticGenerateTest()

```
Require:
  nbG : {nombre d'itérations}
  szP : {taille de population}
  percentOfDeletion : {% de suppression}
  percentOfMutation : {% de mutation}
  mutationVar{} : {probabilité à muté}
  timeOut : {timeout}
  db : {base source contenant les ItemTestPatterns}
  p : {nombre d'ItemTests}
  x : {nombre de choix de réponses}
  m : {nombre de Tests}
  P{} : {population de collection}
  c{} : {une collection}
  i : {compteur}
Ensure:
  {———— Génération de la première population ————-}
  repeat
    c ← randomGenerateTest(db, m, p, x)
    P ← P ∪ {c}
  until P = szP
  i ← 0
  while i < nbG or !timeOut do
    for all c ∈ P do
      averageDtest(c) {la fonstion de FITNESS calcule la distance moyenne entre chaque
      couple de Tests}
    end for
    P.sort()
    {—— suppression du pourcentage prédéfini % and croisement pour compléter la
    population ——}
    deletion(P, percentOfDeletion)
    repeat
      P ← P ∪ {crossover(c1, c2)}
      P.shuffle()
    until #p = szP
    {————————- Mutation du % prédéfini une fois sur trois ————————}
    mutationVar ← range(0, 1000)
    mutationVar.shuffle()
    if mutationVar[0] > 300 then
      mutation(P, percentOfMutation)
      repeat
        c ← randomGenerateTest(db, m, p, x)
        P ← P ∪ {c}
      until #P = szP
    end if
    i ← i + 1
  end while
  P.sort()
  c ← max(P) {la collection la plus différenciée}
  return c
```

4.2.2 Expérimentations et résultats

L'algorithme Génétique présenté à travers l'algorithme 5 a été évalué avec une population de 1000 individus, chacun d'entre eux constituant initialement une *Collection* de *Tests* généré aléatoirement d'une base source d'*ItemTestPatterns*. Relative à cet algorithme, soient $averageDtest(c)$, une fonction qui retourne la distance moyenne d'une *Collection* de *Tests c*, $deletion(P, percentOfDeletion)$ qui supprime $percentOfDeletion$ individus de la population P et $mutation(P, percentOfMutation)$ qui s'occupe de faire la mutation. Les pourcentages de suppression et de mutation sont respectivement fixés de façon empirique à 8% et 3%. Nous considérons 1000 générations (itérations) et un délai maximal d'exécution de 3600 secondes. Pour les paramètres *X1*, *X2*, *X3 et X4*, ils sont fixés afin de ne prendre en compte que de petits ratios (≤ 3) car la génération aléatoire a déjà démontré une performance de différenciation suffisante dans des ratios plus élevés. Ainsi, ces paramètres sont fixés comme suit :

- $X1$ représente la taille de la base d'*ItemTestPatterns*. Cette taille correspond à la quantité d'*ItemTestPatterns* dont dispose l'enseignant dans sa base de données. Ce paramètre est fixé à 40.
- $X2$ représente la taille du *Patron de Collection* de *Tests* à générer ; Cette taille correspond à la quantité de *Tests* que l'enseignant souhaite générer. Ce paramètre varie sur seulement trois valeurs *[2, 3, 5, 7, 10, 15, 30]*.
- $X3$ représente la taille de chaque *Test*. Cette taille correspond à la quantité d'*ItemTests* que l'enseignant souhaite avoir dans chaque *Test*. Ce paramètre varie de *10 à 40*
- $X4$ représente la quantité de *RepItemIts* par *ItemTest*. Donc, la quantité de choix pour chaque *ItemTest* qui est fixé à 4.

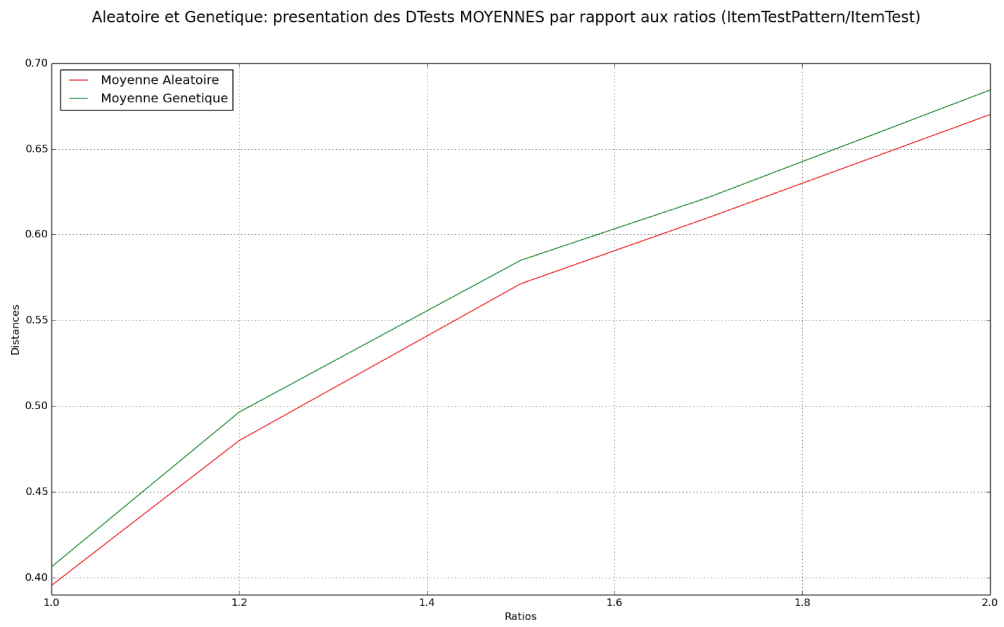

FIGURE 4.9 – Distance moyenne de l'algorithme aléatoire par rapport à celle de l'algorithme génétique

La première remarque sur la contribution expérimentale de méthode de génération avec l'algorithme génétique par rapport à l'aléatoire est présentée dans la figure 4.9. Elle montre une comparaison des distances moyennes que procurent les deux méthodes pour un même ratio. La méthode de génération Génétique garantit à l'enseignant que s'il dispose *n ItemTestPatterns* dans la base de données pour construire des *Tests* de *p ItemTests* pour (n=p, c'est-à-dire ratio = 1,00), une distance moyenne dans le *Patron de Collection* générée de 10% meilleure par rapport à l'Aléatoire. De plus, la figure 4.10 illustre que non seulement la distance moyenne dans le *Patron de Collection* généré avec la méthode génétique est supérieure de 10% à celle de l'Aléatoire, mais c'est également le cas pour les distances minimales et maximales qui dépassent celle de l'Aléatoire du presque même pourcentage.

La figure 4.11 nous permet de regarder les *Patrons de Collections* d'une façon plus locale. Elle représente la distribution des distances pour chaque couple de *Tests* appartenant à la *Collection* 4.9, avec un ratio égal à 1. Grâce à cette figure, nous pouvons observer une comparaison des distances de tous les couples de *Tests* dans un *Patron de Collection* qui a été généré par les deux méthodes. Celle de l'algorithme génétique procure une suite de distances entre les couples de *Tests* largement supérieure à celle de l'algorithme aléatoire. Plus précisément, 60% des distances provenant de l'algorithme aléatoire est inférieure à 0.4, pourtant seulement 30% de celles provenant de l'algorithme génétique

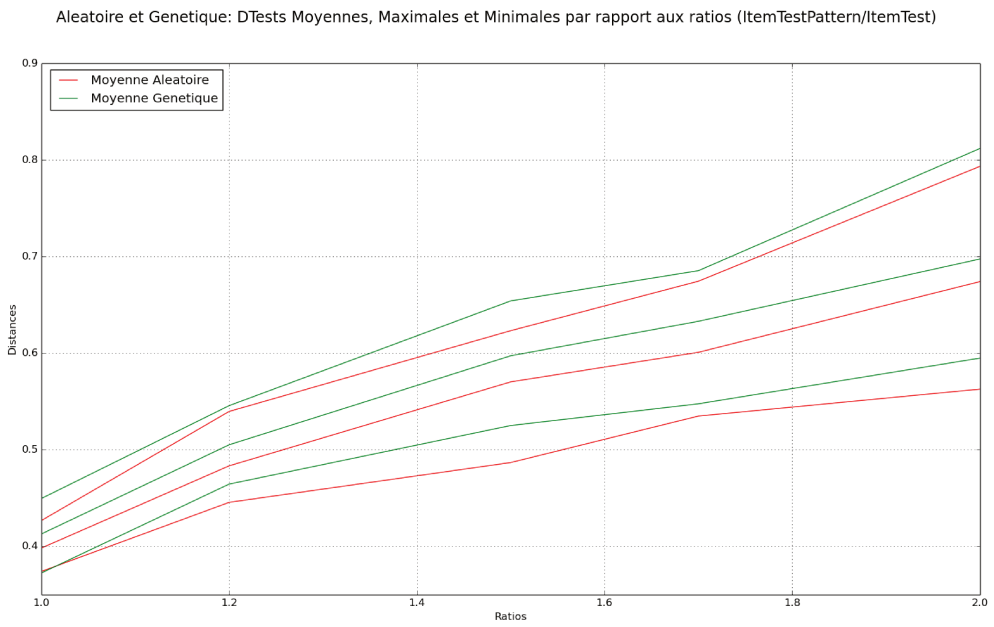

FIGURE 4.10 – Distance moyenne, maximale et minimale de l'algorithme aléatoire par rapport à celle de l'algorithme génétique

sont inférieures à 0.4.

4.3 Bilan et analyse croisée pour les trois méthodes de génération

Les expérimentations menées sur les trois méthodes de génération ont permis de mesurer les limites de performance de trois algorithmes de génération de *Patrons de Collections* de *Tests* différenciés. Nous entendons par performance, leur capacité à générer des *Patrons de Collections* de *Tests* en moyenne très différenciés. Comme spécifié dans la section 3.3.2 du chapitre 3, nous rappelons qu'un *Patron de Collection* de *Tests* est formé que d'identifiants. C'est-à-dire que les *Tests* ne sont constitués que d'identifiants d'énoncés et de choix de réponses. L'avantage est qu'une fois généré, ce *Patron de Collection* devient générique parce qu'il est utilisable pour différentes épreuves d'évaluation dans différentes filières. Pour cela, il faut juste allouer à chaque identifiant un nouveau contenu.

La méthode de génération aléatoire présentée dans le chapitre précédent est considérée comme une méthode de référence. En termes de performance globale, les *Patrons de Collections* qu'elle génère commencent à avoir une différenciation moyenne de plus de 0.8 à partir de ratios supérieurs à trois (3). Par conséquent, l'enseignant doit disposer d'une base source contenant trois fois plus de contenus que ce qu'il veut avoir dans ses *Tests*. Par

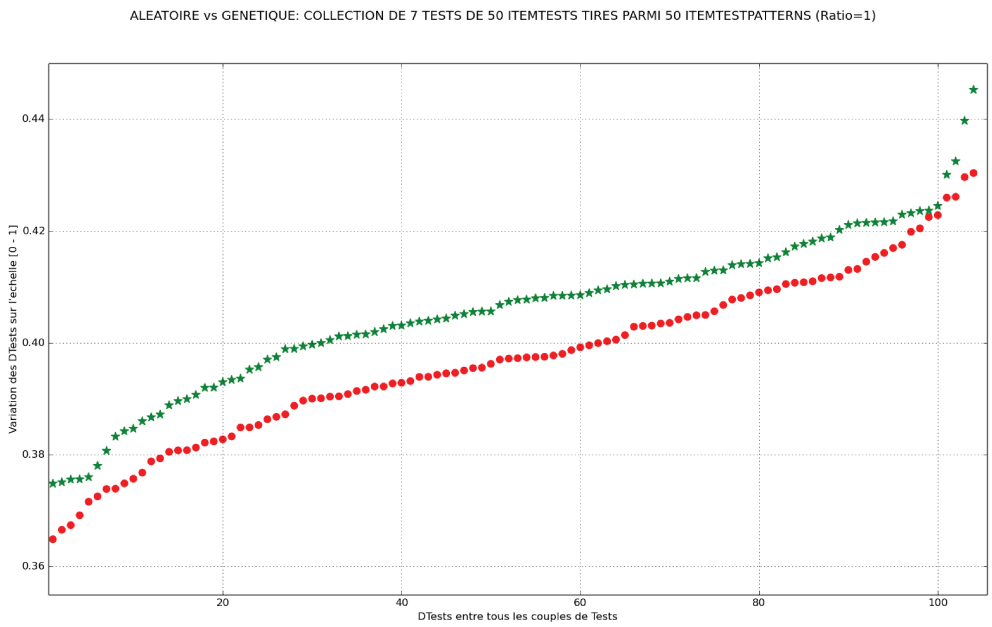

FIGURE 4.11 – Distance moyenne de l'algorithme Aléatoire par rapport à celle de l'algorithme génétique

exemple, un *Patron de Collection* de *Tests* à 60 questions par *Test* doit être construit depuis une base source d'au moins 180 éléments, si l'enseignant espère avoir une différenciation moyenne d'au moins 0.8.

Dans ce chapitre-ci, deux nouvelles méthodes de génération basées sur différentes approches algorithmiques sont étudiées, développées et expérimentées. La première est basée sur un algorithme de Sélection (version HARD et version SOFT). Par rapport à l'aléatoire, on a observé une amélioration de la différenciation moyenne de 5%. La deuxième approche est basée sur un algorithme génétique et les résultats révèlent une meilleure différentiation moyenne par rapport aux deux précédents algorithmes, soit 10% de gain par rapport à l'aléatoire, donc 5% par rapport à la Sélection. La figure 4.12 présente quelques distances moyennes par rapport aux ratios allant de 1 à 3 et pour les trois méthodes de génération.

	DTest<0,5		0.5<DTest<0,7				DTest>0,7	
Ratios	**1**	**1.2**	**1.5**	**1.7**	**2**	**2.1**	**2.5**	**3**
Aléatoire	0.395	0.479	0.571	0.609	0.670	...	>0.700	...
Sélection	0.399	0.472	0.575	0.610	0.69	>0.700	...	...
Génétique	0.406	0.500	0.585	0.621	>0.700	...	...	...

FIGURE 4.12 – Comparaison de la différenciation moyenne pour les trois méthodes de génération

Plus précisément, ce tableau montre une comparaison des limites de chacune des méthodes de génération en termes de différenciation par la présentation d'un ensemble de distances moyennes dans des *Patrons de Collections* de *Tests* générés avec chacune d'elles. On remarque que les méthodes Aléatoire et Sélection procurent une distance moyenne supérieure à 0.5 qu'à partir d'un ratio de 1.5, contrairement à la méthode génétique qui procure une distance moyenne de 0.5 à partir d'un ratio plus faible (1.2). Aussi, on peut remarquer que l'Aléatoire procure une distance moyenne de 0.7 à partir d'un ratio de 2.5, pourtant la méthode de Sélection le procure pour un ratio 2.1 et la méthode génétique procure cette même distance moyenne depuis un ratio encore plus faible. Ainsi, on peut déduire que la méthode génétique est plus performante que les deux autres et que la méthode de Sélection est plus performante que l'Aléatoire.

Regardant les *Patrons de Collections* générés plus localement, c'est à dire les distances entre chaque couple de *Tests* d'un *Patron de Collection*, on trouve une meilleure satisfaction. Pour l'expliquer, la méthode Aléatoire procure de la différenciation moyenne de plus de 0.8 dans les ratios supérieurs à 3, mais on retrouve dans le même *Patron de collection* à la fois des couples de *Tests* très distants et d'autres très proches. Par contre, ceci n'est pas le cas pour la Sélection qui fait en sorte tous les couples de *Tests* n'aient de distances inférieures au seuil ou à l'intervalle de seuils qui est prédéfini. L'algorithme génétique pour sa part réduit aussi l'écartype, mais pas de la même façon que la Sélection. La méthode génétique tente de maximiser la différenciation entre tous les couples de *Tests* en les faisant remonter le plus haut possible en même temps, alors que la Sélection écrête les distances minimales. Ainsi, les méthodes de génération par la Sélection et la méthode génétique font en sorte que la distance entre tous les *Tests* deux à deux soient supérieure à

un certain seuil.

Ceci dit, dans l'hypothèse d'une utilisation réelle du générateur DIFAIRT-G, deux principaux cas de figures peuvent se présenter. Le premier est celui où la *Collection de Tests* se génère directement par rapport aux paramètres rentrés par l'enseignant et le deuxième cas correspond à celui où la *Collection de Tests* se génère en s'appuyant sur un *Patron de Collection de Tests* existant. Ce dernier cas suppose qu'une génération aux paramètres similaires a déjà été faite, donc il n'y a qu'à rajouter des contenus aux identifiants des énoncés et des choix de réponses.

Dans le premier cas de figure, pour utiliser la méthode de génération optimale en termes de différenciation, DIFAIRT-G opère le calcul du ratio. Par exemple, sachant que $X1 = 60$, $X2 = 100$, $X3 = 10$ et $X4 = 4$ représentent respectivement les paramètres de la taille de la base source, la taille de la *Collection de Tests* à générer, le nombre de questions par *Test* et le nombre de choix de réponse par question, le ratio vaut 6. Ainsi, c'est la méthode de génération Aléatoire qui sera utilisée, car elle garantira une différenciation moyenne supérieure à 0.8 pour ce ratio. Par contre, si on change la valeur du paramètre $X3$ en $X3 = 30$, le ratio sera égal à 2. Alors, les méthodes de génération par Sélection et Génétique sont les meilleures candidates. Toutefois, pour choisir entre les deux, DIFAIRT-G vérifie si l'enseignant avait précisé une différenciation seuil. Si oui, la Sélection convient au mieux car elle permet de filtrer les *Tests* générés dès leur création. Sinon, c'est la méthode génétique qui convient le mieux. Ce premier cas de figure est résumé dans le tableau 4.1.

CHOIX	**CONDITIONS**
Méthode Aléatoire	Si ratio > 3
Méthode Sélection	Si ratio < 3 et il existe un seuil prédéfini
Méthode Génétique	Si ratio < 3

Tableau 4.1 – Résumé du premier cas de figure de génération

Dans le deuxième cas de figure, si on se retrouve face à une configuration similaire qui a déjà été traitée, DIFAIRT-G peut tout simplement générer la *Collection* de l'enseignant en s'appuyant sur un *Patron de Collection* existant. Dans le cas où les deux configurations seraient légèrement différentes, DIFAIRT-G propose à l'enseignant de retoucher sa configuration, comme procéder à l'ajout de quelques questions et/ou choix de réponses dans sa base pour mieux se synchroniser avec le *Patron de Collection* existant. Par exemple, supposant que pour des paramètres $X1 = 60$, $X2 = 100$, $X3 = 30$ et $X4 = 4$ DIFAIRT-G avait déjà généré un *Patron de Collection z* d'une différenciation moyenne de 0.7 ; et que l'enseignant dispose de presque les mêmes paramètres sauf que son $X3 = 25$, une possibilité est qu'il

rajoute 5 nouvelles questions dans sa base pour avoir $X3 = 30$, ce qui lui rassure une différenciation minimale de 0.7 dans sa *Collection* qui sera générée depuis le *patron de Collection z*. Ce deuxième cas de figure est résumé dans le tableau 4.2.

CHOIX	CONDITIONS
Option 1	Tous les paramètres correspondent aux caractéristiques d'un patron de collection existant, génération directement
Option 2	Les paramètres sont proches des caractéristiques d'un patron existant, l'enseignant réadapte ses paramètres

Tableau 4.2 – Résumé du deuxième cas de figure de génération

Il est à retenir que les trois méthodes trouvent leur importance selon les contextes de génération. C'est à dire, l'algorithme aléatoire peut mieux convenir à un enseignant qui dispose d'une grande taille de base source ou qui ne dispose pas de beaucoup de temps pour faire sa génération. En d'autres mots, si l'enseignant dispose de trois fois plus d'*ItemtestPatterns* que d'*ItemTests*, ou s'il veut impérativement générer son *Patron de Collection* en quelques secondes, la méthode de génération Aléatoire peut être un très bon candidat. D'un autre coté, la Sélection conviendra mieux à un enseignant qui tient absolument à ce que tous les couples *Tests* de son *Patron de Collection*, aient un niveau de différenciation minimal. L'algorithme génétique pour sa part conviendrait le mieux à celui qui dispose d'une petite taille de base, mais qui veut avoir une différenciation au mieux possible. On notera que dans les trois méthodes, le fait d'augmenter la quantité de choix dans la base peut garantir une grande différence.

4.4 Prototypage actuel de DIFAIR-G

Cette section a pour objectif de présenter le prototype existant. Nous présentons le coté technique de DIFAIRT-G en détaillant l'architecture actuelle et la façon dont les *Patrons de Collection* sont générés.

4.4.1 Architecture actuelle

Pour le moment, l'architecture du prototype est adaptée à une utilisation sur un poste de travail fixe. Cela suppose que l'utilisateur dispose du code source du générateur sur son ordinateur. Si on se réfère à la figure 4.13, on voit que l'accès et l'utilisation du générateur se fait en ligne de commande. La première étape consiste au paramétrage du générateur en renseignant les informations requises comme la taille de la collection à générer, la source des *ItemTestPatterns*, la quantité de questions et de choix de réponse par *Test*, etc. L'étape 2 est celle de la génération des *Tests*. Cette étape ne demande aucune assistance de l'utilisateur. L'étape 3 consiste en la création et le stockage des *Tests* générés. Aussi, un fichier historique est généré pour chaque *Patron de Collection*. La dernière étape vient

après la composition. Il s'agit de la correction automatique des copies des apprenants après la numérisation de ces dernières.

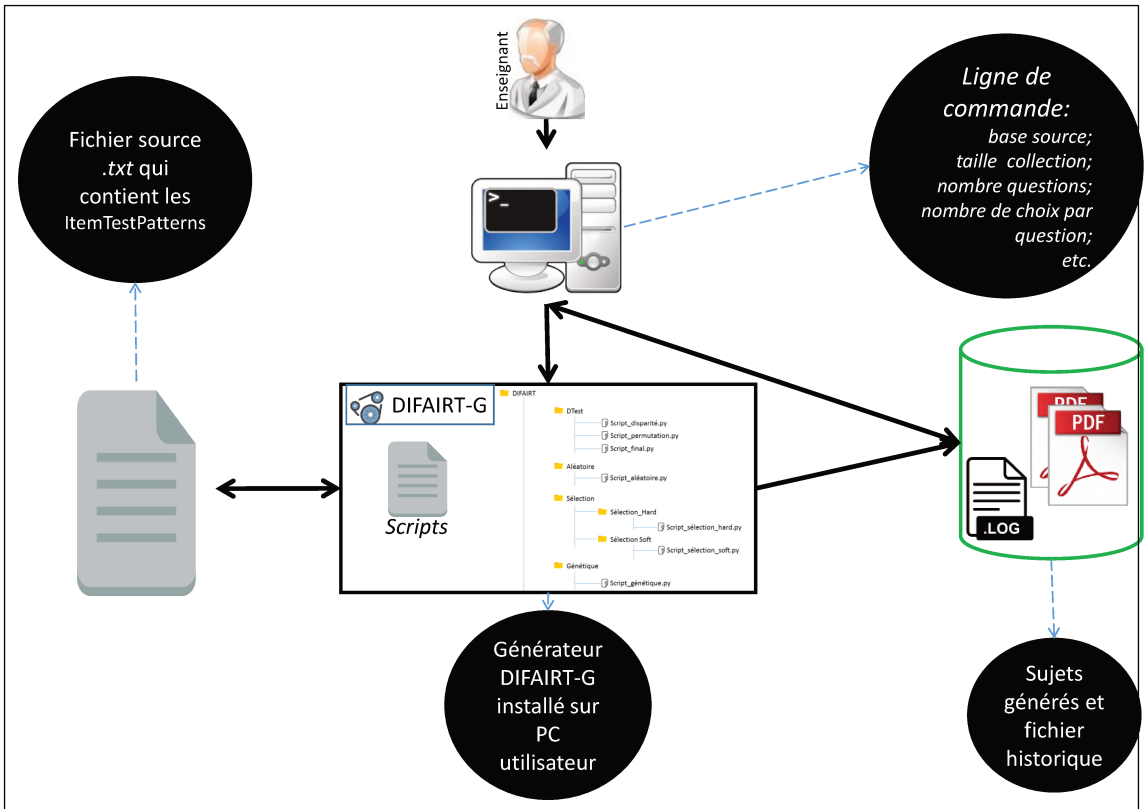

FIGURE 4.13 – Architecture actuelle

Le prototype est développé sur la distribution Ubuntu de Linux [1], en Python [2]. Le choix de Python est fait pour son aisance dans la manipulation des chaînes de caractères et parce que YMCQ (l'ancêtre de DIFAIRT) est également en Python, ce qui permet de garder une compatibilité entre les deux. Les scripts de DIFAIRT sont classés dans un même dossier nommé *DIFAIRT* et les principaux sont les suivants :

- Scripts *DTest()* : subdivisés en un script permettant le calcul de la disparité entre deux *Tests* ; un script permettant le calcul de la permutation entre les éléments communs de deux *Tests* ; et un script permettant le calcul final de *DTest()*.
- Script génération Aléatoire : permettant de gérer le tirage aléatoire des énoncés et choix de réponse pour construire chaque *Test* du *Patron de Collection*.
- Scripts génération Sélection : subdivisés en deux scripts dont le premier qui permet de faire la génération de *Patrons de Collections* par la méthode de la Sélection HARD, et le deuxième par la Sélection SOFT.

1. Linux : noyau de système d'exploitation
2. Python : Langage de programmation interprété et multiplateformes favorisant la programmation fonctionnelle et orientée objet

- Script génération Méta-heuristique : permettant la génération de *Patrons de Collections* par la méthode génétique.

La section suivante présente la démarche globale de la génération et quelques exemple de *Patrons de Collections*.

4.4.2 Génération des *patrons de Collections*

Comme il est présenté dans ces deux derniers chapitres, pour mesurer la différenciation entre deux *Tests*, la métrique *DTest()* ne considère que les identifiants des énoncés et des choix de réponse. Nous rappelons que le principal avantage est qu'une fois qu'un *Patron de Collection* de *Test* est généré, il pourra être réutilisé lors de différentes épreuves d'évaluation car seulement les contenus des énoncés et des choix de réponses changeront.

Pour générer un *Patron de Collection*, on fait le choix de la méthode de génération, tout en la paramétrant. Si c'est la première méthode qui est choisie, le *Patron de Collection* est généré avec l'algorithme Aléatoire. La distance entre tous les couples de *Tests* est calculée ainsi que la distance moyenne. Ces calculs sont faits à but informatif car ils ne conditionnent pas la validation des *Tests* générés. Si c'est la deuxième méthode qui est lancée, c'est l'algorithme de génération par la Sélection qui est utilisé. A la génération de chaque *Test*, sa distance moyenne est calculée avec les autres générés avant lui pour être comparée au seuil prédéfini. Ce dernier est accepté et ajouté dans le *Patron de Collection* s'il valide la condition (*DTest>Seuil*), sinon il est détruit. Ce processus continue jusqu'à la complétion du *Patron de Collection* avec la quantité de *Tests* souhaitée. Quant à la troisième méthode, c'est celle de l'algorithme génétique. La génération de multiple *Patrons de Collections* est faite, ces derniers sont ensuite triés des plus faiblement distants aux plus fortement distants. Les opérations génétiques (suppression, mutation et complétion) sont faites d'itération en itération. A la dernière itération, le *Patron de Collection* ayant la distance moyenne la plus forte est sauvegardé. La section qui suit présente quelques travaux en cours sur la mise en place d'une interface utilisateur.

4.5 Vers une interface AMI-DIFAIRT

Cette section a pour objectif d'introduire l'interface utilisateur AMI-DIFAIRT (Assessment Management Interface - DIFAIRT). Il s'agit d'une interface de pilotage dédiée à l'enseignant utilisateur de DIFAIRT. Il a pour objectif de rendre plus fluide les interactions des enseignants avec DIFAIRT en facilitant l'accès, l'organisation et la constitution des bases sources, mais aussi en procurant une assistance au lancement des différents algorithmes de génération, le tout sur une architecture client-serveur. A terme, AMI-DIFAIRT doit également permettre un accès aux fonctionnalités de corrections et d'analyse des

résultats. L'interface principale est un portail qui réunit toutes les fonctionnalités de DIFAIRT à la fois. Plus précisément, on y trouve des fonctionnalités de constitution/gestion de la base source en offrant la possibilité à l'enseignant de rentrer ses questions et réponses manuellement ou de les importer depuis d'autres systèmes de gestion des évaluations. On y retrouve aussi les fonctionnalités de génération et de correction de *Tests*.

4.5.1 Architecture nouvelle

L'interface AMI-DIFAIRT a exigé une modification de l'architecture qui est actuellement en place. Le changement majeur entre les deux est l'utilisation de l'interface graphique à la place des lignes de commande. Se référant à la figure 4.14, on peut voir que la première étape est subdivisée en 3 sous-étapes dont la connexion de l'utilisateur sur son compte, le paramétrage de la génération et le lancement de la génération. La base source devient une base de données (MySQL pour le moment). La génération se fait depuis un serveur distant et les *Tests* peuvent être téléchargés sur un format PDF après la génération. La phase de la correction ne change pas. A terme, un module d'analyse statistique et sémantique des résultats d'évaluation doit être ajouté à l'architecture pour d'une part donner des rapports à l'enseignant utilisateur et d'autre part pour enrichir la base source.

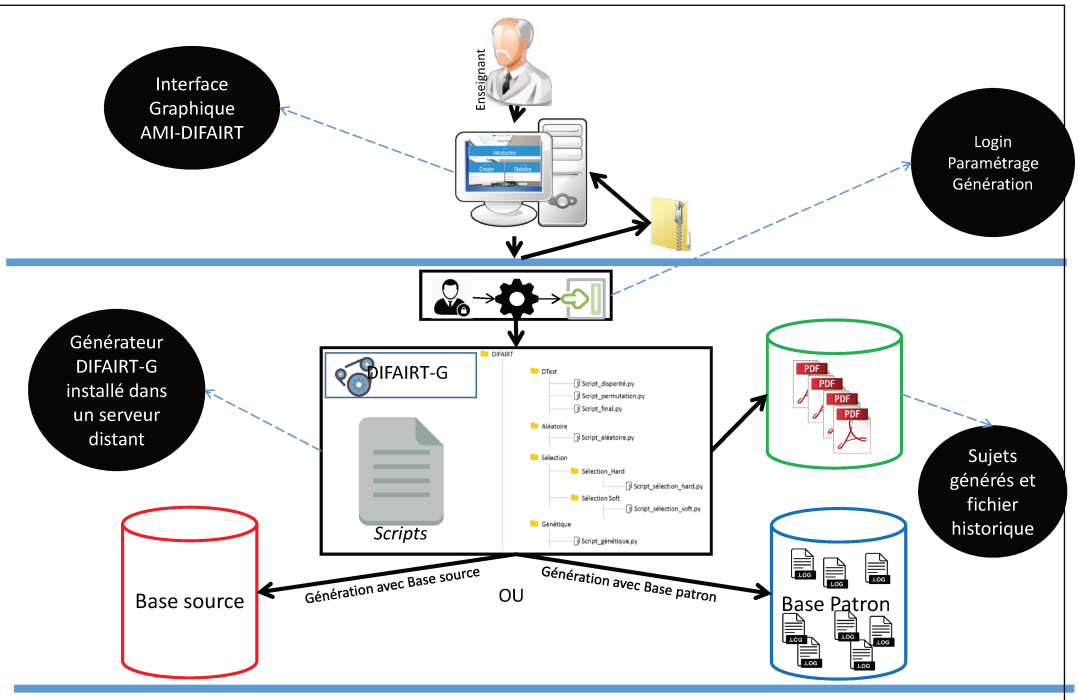

FIGURE 4.14 – Architecture AMI-DIFAIRT

4.5.2 Paramétrage et génération avec AMI-DIFAIRT

Ainsi, pour générer un *Patron de Collection* de *Tests*, il faut impérativement lui fournir un minimum d'informations, ce sont les paramètres de génération. Dans cette nouvelle architecture, deux options de paramétrage sont proposées à l'enseignant : le mode simple paramétrage (appelé aussi génération assistée) où l'enseignant ne fournit que les paramètres de base ; et le mode paramétrage avancé (appelé aussi génération personnalisée) qui permet de paramétrer jusque dans les détails les caractéristiques de son *Patron de Collection*. L'interface de la gestion du paramétrage et de la génération est schématiquement représentée dans la figure 4.15.

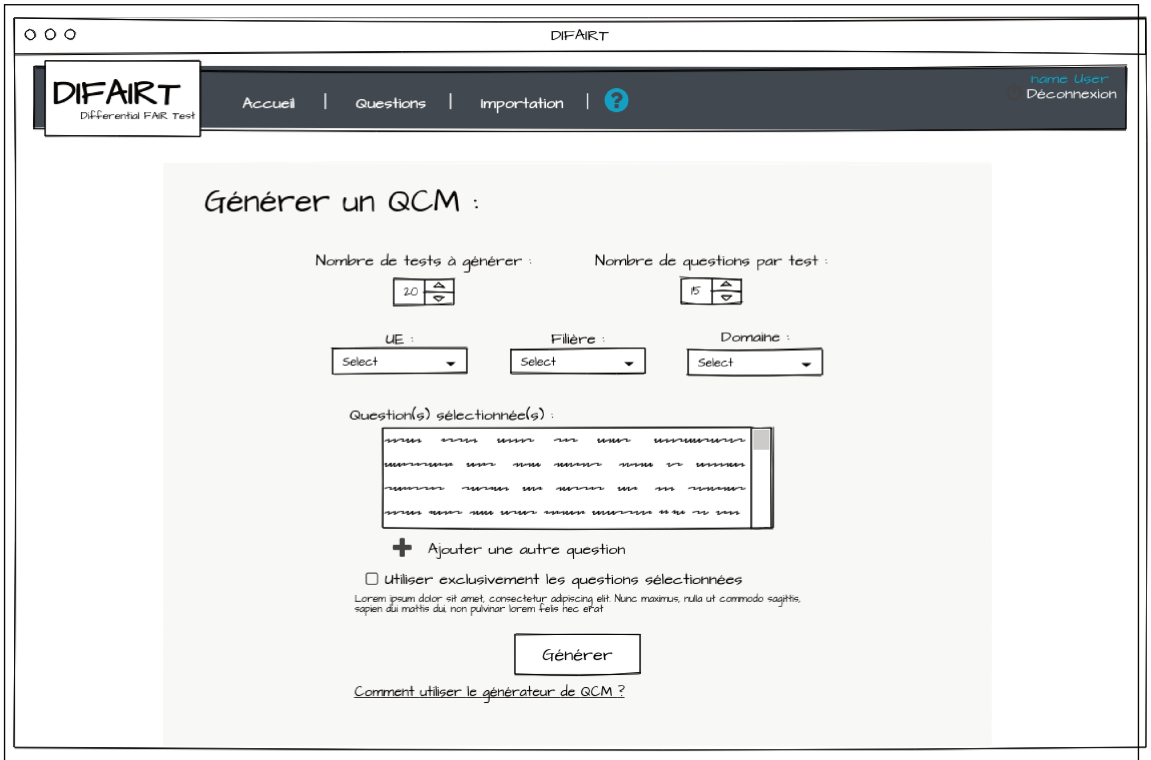

FIGURE 4.15 – Paramétrage de la génération

Dans le premier cas, l'enseignant doit spécifier : la quantité de *Tests* à générer ; la quantité de questions par *Test* ; et la quantité de choix par question. Ainsi, il revient à DIFAIRT-G de trouver la méthode de génération la mieux adaptée en termes de différenciation pour créer le *Patron de Collection* de l'enseignant. Cette génération dépend exclusivement des paramètres de génération pour choisir la méthode. Par exemple :

- Prenant connaissance de la quantité de questions à mettre dans les *Tests* par rapport à la quantité de questions dans la base source, DIFAIRT-G connaît le ratio d'*ItemTestPattern* par *ItemTest*. Ainsi, si le ratio est supérieur à 3, l'aléatoire devrait

suffire pour procurer une assez grande différenciation moyenne. Mais dans le cas où le ratio serait inférieur ou égale à 3, DIFAIRT-G sait que la différenciation est meilleure avec les méthodes de Sélection et Génétique.

- Grâce à la quantité de choix spécifiée par l'enseignant, DIFAIRT-G connaît le rapport entre la quantité de choix dans la base source et la quantité de choix dans les questions du *Test* (ratio des choix *RepItemITP/RepItemIT*). Cette information joue un rôle important dans la génération même si le ratio des énoncés est inférieur à 3, celui des choix peut permettre une augmentation considérable de la différenciation moyenne dans le *Patron de Collection* à générer.

Pour le cas du mode paramétrage avancée, l'enseignant spécifie les paramètres additionnels comme : le temps maximal de génération (donc il fixe un timeout en raison de ses contraintes de temps) ; le niveau de difficulté souhaité dans son *Patron de Collection*, ce qui fera en sorte que tous les *Tests* aient un niveau de difficulté plus ou moins équivalent ; la différenciation moyenne souhaité en fixant un seuil allant de 0 à 1. C'est à dire qu'aucun couple de *Tests* dans son *Patron de Collection* n'aura un niveau de différenciation inférieur à ce seuil ; il peut même choisir la méthode de génération qu'il souhaite utiliser directement. Toutefois, fait de paramétrer lui-même la méthode de génération peut supposer qu'il a des connaissances techniques assez poussées, lui permettant de savoir au préalable quel type d'algorithme convient au mieux avec ses paramètres.

Le lancement de la génération avec ce mode de paramétrage est plus simple à gérer pour DIFAIRT-G parce-qu'il ne fait pas forcément de calculs en amont de la génération, mais le risque du non-aboutissement de la génération est plus élevé. Ceci s'explique par le fait qu'en choisissant l'option de tout paramétrer soi-même, l'enseignant peut avoir été trop exigeant sur les contraintes de différenciation et de temps de génération. Toutefois, des consignes d'assistance seront disponibles pour aider l'enseignant le paramétrage avancé de son générateur.

4.5.3 Constitution de la base source et navigation

D'autres fonctionnalités sont implémentées dans l'interface AMI-DIFAIRT comme celle de la création de la base des *ItemTestPatterns* où l'enseignant peut soit importer ses contenus, soit les saisir manuellement. La figure 4.16 illustre cette fonctionnalité. Ces données peuvent provenir de n'importe quel autre systèmes que l'enseignant utilisait avant. Toutefois, seulement les fichiers sources *.txt* provenant de YMCQ peuvent être importés pour le moment. A terme, on veut pouvoir importer d'autres types de fichiers sources comme *.doc*, soit *.xls* soit *.txt*. si ils respectent la structure de données de DIFAIRT. Un script spécifique vérifie le type de fichier et la structure de son contenu avant de l'insérer dans la base. Aussi, AMI-DIFAIRT permet d'exporter ses données vers d'autres

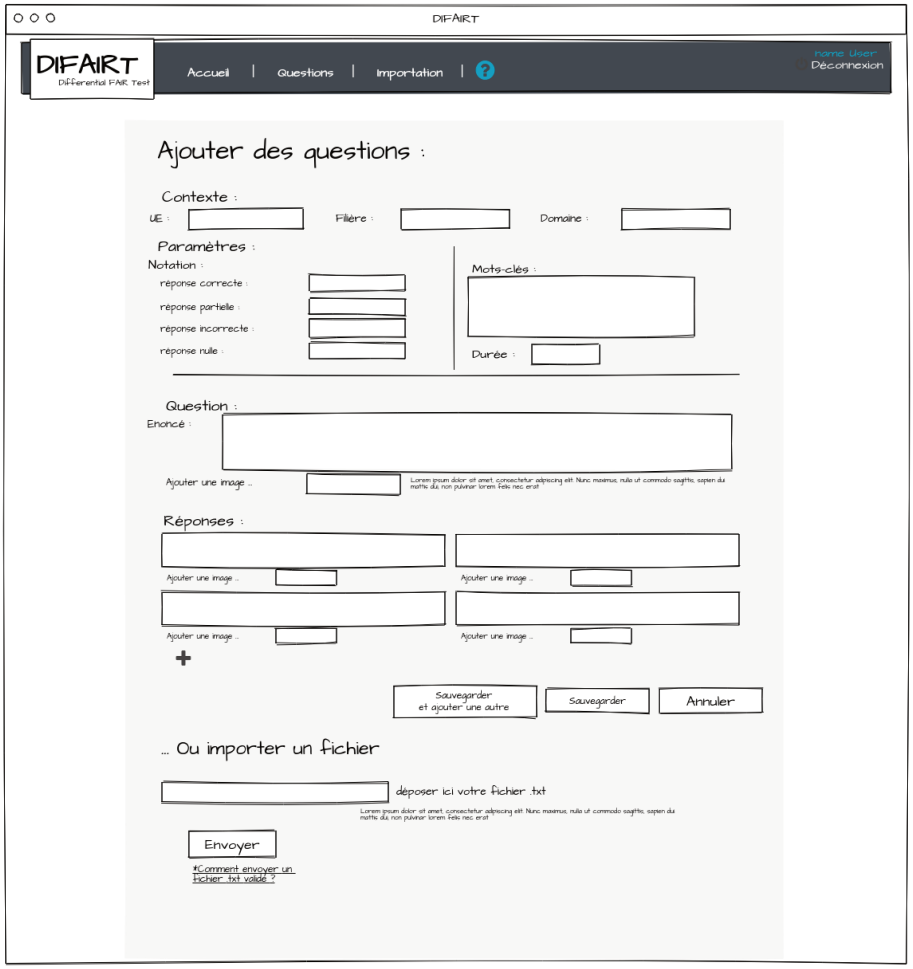

FIGURE 4.16 – Interface d'importation et d'exportation des questions

systèmes. Cela se fait aussi sous des formats bien déterminé, respectant les normes de compatibilité avec les systèmes respectant QTI. Les types de fichiers exportés sont les *.xls* et *.txt*.

En termes de navigation et visualisation des questions et réponses de la base source, Comme il est illustré dans la figure 4.17, l'enseignant peut apporter directement des modifications dans ces contenus. Par exemple, déterminer les bonnes ou/et mauvaises réponses, éditer les énoncés ou encore supprimer des éléments.

Depuis cette même fonctionnalité, certains outils permettent à l'enseignant d'effectuer des recherches de contenus spécifiques. Car au-dessus d'une certaine quantité d'éléments, il devient plus compliqué de naviguer sur cette page. La page est aussi munie de filtres, permettant de trier les éléments par rapport à certains critères précis comme par Unité d'Enseignement, par Domaines ou encore par Filière.

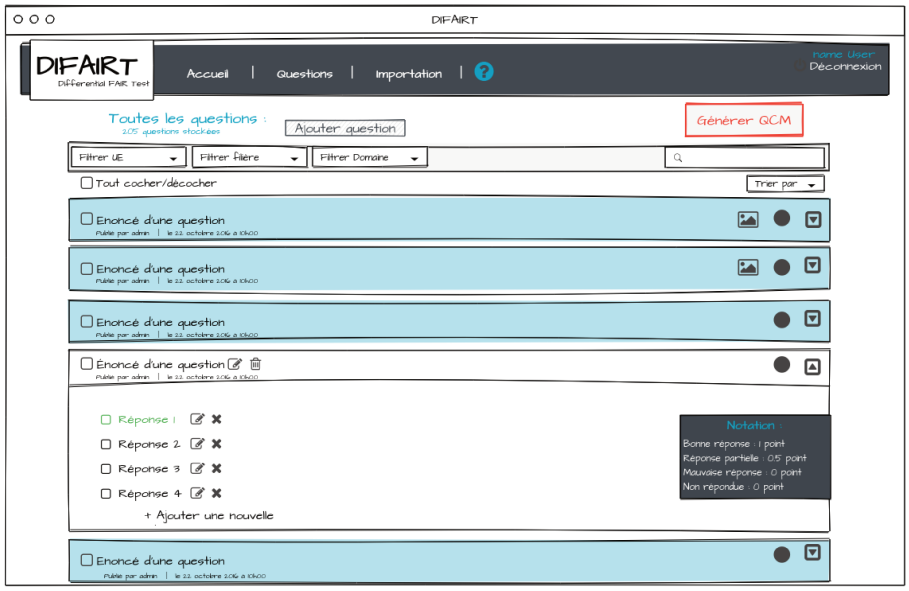

FIGURE 4.17 – Interface de visualisation et de modifications

4.5.4 La correction automatique dans AMI-DIFAIRT

La phase de la correction commence juste après la composition des étudiants. Ce que nous présentons dans la suite de cette section suppose que la composition est faite sur des supports papier. Ainsi le correcteur de DIFAIRT est basé sur la méthode de correction de YMCQ. On entend par là, la numérisation des copies des étudiants, correction des versions numérisées grâce à versions corrigées de DIFAIRT, l'attribution des notes et finalement la visualisation des résultats statistiques. Le processus de correction automatique débute après la numérisation des copies.

4.6 Questionnements de transition au prochain chapitre

Se référant à la synthèse que présente la figure 4.12 dans la section 4.3, on se demande si on peut faire mieux en termes de différenciation dans un contexte d'évaluation certificative à l'université. Existe-t-il une autre méthode qui pourrait garantir une épreuve d'évaluation où les apprenants auraient tous des *Tests* plus différenciés que ce que proposent nos trois algorithmes ? Sachant que les distances moyennes dans les grands *Patrons de Collections* générés jusque-là ne sont pas forcément significatives, cela serait-il différent si on générait des Patrons de *Collections* de taille plus petite ?

Dans le prochain chapitre nous proposons de ne plus réfléchir en différenciation moyenne sur des grands *Patrons de Collections*, mais de considérer l'hypothèse de la

génération des *Patrons de Collections* de taille réduite pour espérer de meilleures différenciations moyennes.

Chapitre 5

Vers une optimisation de la différenciation : le cas des petites Collections

Cette partie de nos travaux de recherche fait suite aux deux précédents chapitres qui ont montré que les trois méthodes de génération pouvaient permettre d'avoir une différenciation moyenne sur toute une *Collection* de *Tests* générée, mais faible en moyenne quand la taille de la base source n'est pas assez conséquente. Par exemple, si on se focalise sur les ratios allant de 1 à 3, la différenciation moyenne ne dépasse pas 0.8. Dans ce cadre, les résultats ont démontré que la méthode de génération la moins performante sur les petits ratios en terme de différenciation est celle qui utilise l'algorithme Aléatoire, et la plus performante est l'approche Génétique (soit +10% de gain de performance par rapport à l'Aléatoire). Supposant qu'un enseignant cherche avant tout une grande différenciation dans sa *Collection* mais ne s'attache pas forcément à une différenciation pour chaque étudiant, nous pouvons conclure que cela ne sera pas toujours possible avec ces premières méthodes (dépendamment de la taille de la base source ou encore de la taille de la *Collection*).

Ce chapitre propose une orientation de nos travaux vers la génération de *Patrons de Collections* de plus petites tailles en vue d'obtenir une meilleure différenciation. Il s'agit d'une approche à travers laquelle nous faisons l'usage des graphes, pour la modélisation/visualisation de la différenciation entre les *Tests* générés. De cette approche découle ensuite la problématique d'attribution de petites *Collections* de *Tests* aux étudiants de façon optimale (c'est-à-dire pour limiter les fraudes). Par exemple, si on part de l'hypothèse que l'évaluation aura lieu dans une salle de classe physique, la contrainte principale de l'enseignant est de faire en sorte que les étudiants voisins reçoivent des *Tests* les plus

différenciés possible.

La première section est consacrée à la question de l'attribution des petites *Collections* en salles d'examen de façon optimisée. La deuxième section de ce chapitre présente une première vérification de l'hypothèse de l'augmentation de la différenciation dans des *Patrons de Collections* de taille réduite, à travers les expérimentations passées. La troisième section présente l'approche de l'identification des petites *Collections* très différenciées au sein de plus grandes *Collections*. La dernière section avance quelques perspectives de recherche sur l'approche de génération de petites *Collections* et d'attribution de sujets différenciés en salles d'examen.

5.1 Méthodologie d'attribution de sujets

Le fait de générer des patrons de *Collection* de taille réduite interroge en premier lieu l'attribution des sujets générés aux étudiants visés. En effet, pour une *Collection* de taille réduite il n'y a pas assez de *Tests* différenciés pour que chaque étudiant reçoive un *Test* différent des autres, certains vont se trouver avec un même *Test* (même contenu dans les mêmes placements). Alors, comment faire pour que deux étudiants voisins reçoivent toujours deux *Tests* distants ? Nous supposons que lors d'une épreuve d'examen, le but principal de l'enseignant qui fait la différenciation est de limiter la fraude. Ainsi, il fait en sorte que les étudiants qu'il estime être voisins (exemple : assis cote-à-cote) obtiennent des *Tests* différenciés.

La notion de voisinage est ici relative car elle est considérée et/ou définie par l'enseignant selon ses propres critères. Pour nous, cette notion de voisinage ne se résume pas seulement à deux étudiants assis dans une même salle physique. Elle peut inclure des cas de figures comme celui d'une épreuve d'évaluation en ligne pour un même cours, sur des plages d'horaires différentes ou encore dans des espaces géographiques différents. La suite de cette section est principalement dédiée à la présentation de cette approche d'attribution reposant sur le problème de la coloration d'un graphe.

5.1.1 Représentation du voisinage

Pour pouvoir rendre compte des contraintes de voisinage, nous proposons de construire un graphe. Ainsi, on peut par exemple représenter une salle de classe à l'aide d'un graphe $G_P = (S, A)$ appelé graphe des places où chaque sommet représente une place disponible pour un étudiant. Il existe une arête entre deux sommets si et seulement si l'enseignant estime que les deux places sont "voisines". Ainsi, S est l'ensemble des places d'affectation allant de p_1 à p_n et A={$\exists(p_i, p_j)$} ssi p_i et p_j sont définis comme voisins. Chaque sommet pourrait également représenter une plage d'horaire ou un espace géographique dans

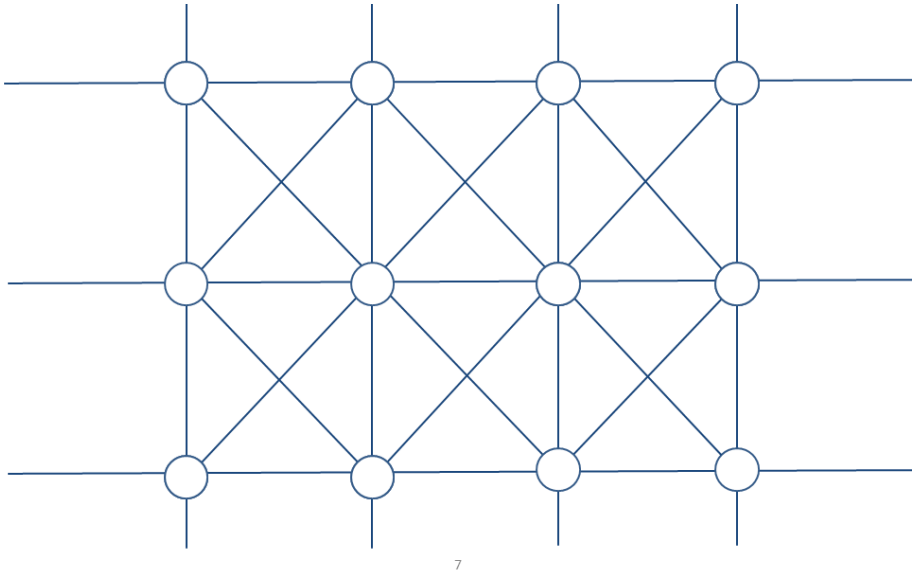

FIGURE 5.1 – Graphe G_P représentant le voisinage dans un amphithéâtre

le cas du passage d'un examen en ligne, voire un étudiant qui ne doit pas avoir un sujet semblables à un tel autre. La figure 5.1 illustre un graphe G_P de l'exemple d'un amphithéâtre où l'enseignant considère que chaque place a huit voisins (avant, arrière, droite, gauche, arrière droit, arrière gauche, avant droit et avant gauche), et donc devant recevoir des *Tests* différents du sien. Il s'agit d'une topologie assez classique rencontrée en contexte universitaire.

5.1.2 Affectation des sujets

Une fois le graphe de voisinage G_P représenté, il nous faut réfléchir à la méthodologie d'affectation des sujets. Ainsi, on se ramène au problème de la coloration d'un graphe qui consiste à attribuer une couleur à chaque sommet d'un graphe de manière que deux sommets reliés par une arête soient de couleurs différentes [34]. Il s'agit de déterminer le nombre de couleurs (nombre chromatique) nécessaire pour colorer le graphe. Il existe deux problèmes de coloration de graphe qui sont : « problème de décision » qui pose la question « existe-t-il une coloration valide d'un graphe G_i utilisant n couleurs ? » et le « problème d'optimisation » qui pose la question « quel est son nombre chromatique ? ». Dans notre cas, le premier problème peut nous permettre de répondre à la question s'il est possible de distribuer n *Tests* sur une topologie de salle spécifique et le deuxième problème peut nous permettre de répondre à la question du nombre minimum de *Tests* différenciés qu'il nous faut pour une distribution sur une topologie de salle spécifique.

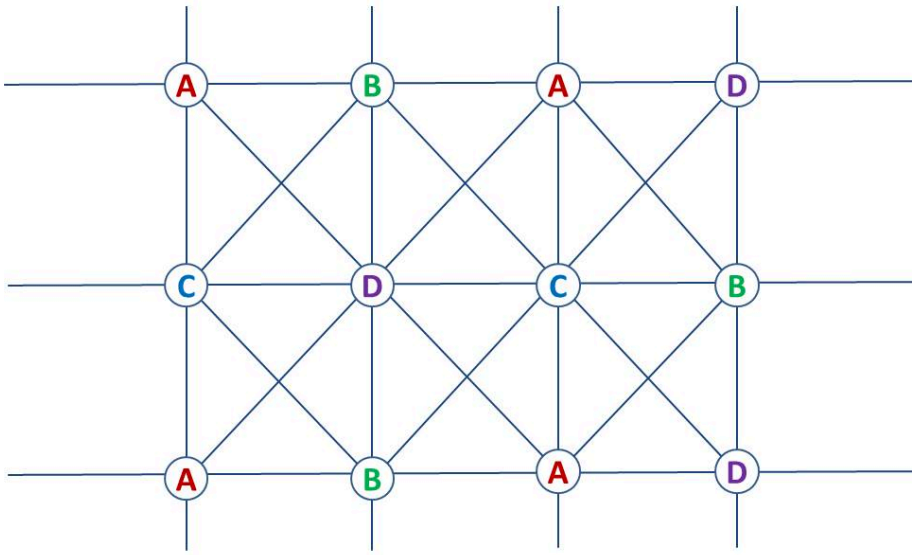

FIGURE 5.2 – Exemple de distribution en suivant l'approche de la coloration de graphe

Considérant l'exemple définit dans la figure 5.1, l'étape suivante consiste alors à déterminer le nombre de *Tests* minimum nécessaires. Ainsi, il apparaît que ce G_P est un graphe dit 4-coloriable [45] c'est à dire qu'il faut au minimum 4 couleurs pour réaliser sa coloration. Donc, si chaque couleur représente un *Test* différencié, il nous faudrait un petite *Collection* contenant au minimum 4 *Tests* fortement différenciés pour effectuer une distribution optimale dans cet amphithéâtre, pour être sûr que chaque apprenant recevra un *Test* très différent de celui de son voisin. Supposons que les couleurs soient nommées A, B, C et D, voici un exemple de coloration du graphe G_P (distribution des *Tests*) représentant les contraintes de voisinage évoquées dans la figure 5.2. Toutefois, nous n'écartons pas la possibilité qu'un enseignant se retrouve dans une salle avec une topologie différente de cette dernière, ce qui serait plus compliqué en termes de coloration comme le graphe quelconque représenté dans la figure 5.3. Pour la suite de nos travaux, nous utilisons le cas de cet amphithéâtre qu'à titre illustratif et expérimental.

Disposant de cette approche de distribution dans les salles basée sur les graphes, maintenant la question est : sommes-nous en mesure de procurer de petites *Collections* de *Tests* très différenciés de taille suffisante pour les distributions ? Si oui, dans quelle configuration (ratio des questions et des réponses) ? Avec quelle méthode de génération ? Dans quelle limite (en termes de différenciation) ?

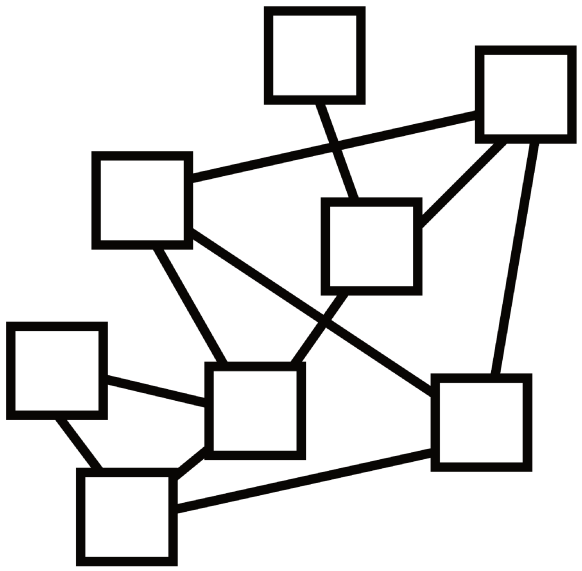

FIGURE 5.3 – Exemple de représentation d'une salle de classe sous forme de graphe quelconque

Avant tout, est-ce vrai que le fait de réduire la taille de la *Collection* à générer permet d'augmenter la différenciation ? La section suivante fait un tour des premières expérimentations pour trouver dans quels cas de figures où la réduction de la taille d'une *Collection* cause une augmentation de sa différenciation moyenne.

5.2 Meilleure différenciation dans les plus petites Collections ?

Pour vérifier la pertinence de cette approche, la question suivante est posée : considérant un même ratio, une même taille de base source et un même nombre d'*ItemTests* dans les *Tests*, est-ce que la taille des *Collections* influe sur la différenciation moyenne ? Autrement dit, si on utilise nos méthodes de générations pour des *Collections* de différentes tailles comme [10, 20, 80 et 160] *Tests*, à partir d'une même base source, avec des *Tests* contenant une même quantité d'*ItemTests*, est-ce que la distance moyenne est plus forte dans la *Collection* la plus petite ?

Pour répondre à cette question, nous avons observé la différenciation moyenne dans nos premières expérimentations faites avec la méthode de génération Aléatoire pour les tailles de *Collection* sus-citées. En résultat, on a remarqué que la distance moyenne varie peu. La figure 5.4 montre la distance moyenne dans les quatre *Collections* de 10, 20, 80 et 160 *Tests* qui stagne aux alentours de 0.45. En d'autres mots, la taille de la *Collection* n'influe pas sur la différenciation moyenne pour un même ratio. Donc, avec la méthode de génération Aléatoire, le fait de réduire la taille des *Collections* ne permet pas d'augmenter

la distance moyenne.

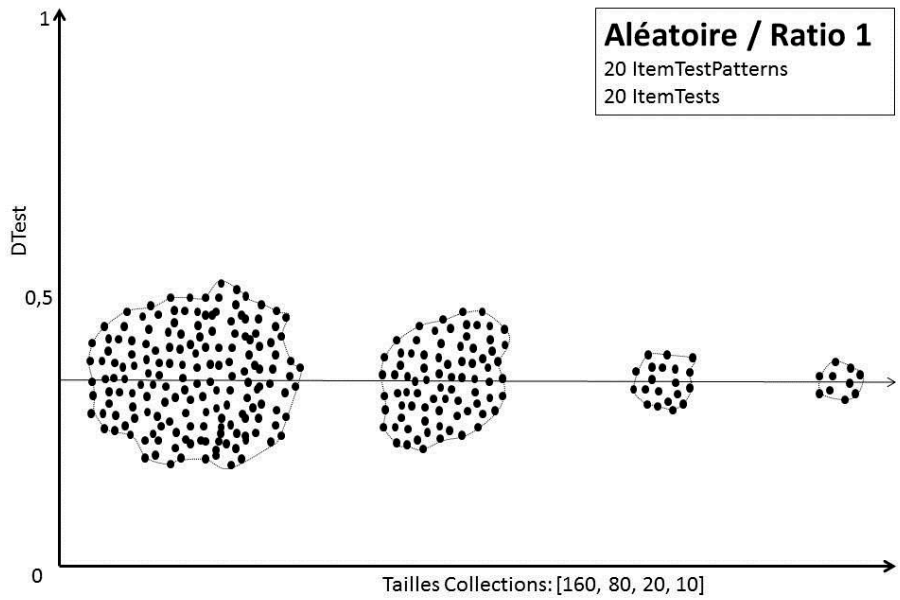

FIGURE 5.4 – La distance moyenne pour des tailles de collection différentes avec l'aléatoire

Le même protocole a été reproduit sur les résultats des expérimentations avec l'approche de génération Génétique. Contrairement à l'Aléatoire, la distance moyenne augmente lorsque la taille de la *Collection* générée est moins importante. La figure 5.5 représente les mêmes *Collections* pour l'approche Génétique et on observe que la distance moyenne croit de 0.41 à 0.5 mais qu'aussi la distance maximale ne dépasse pas 0.5.

Ces observations signifient pour nous qu'il y a la possibilité d'obtenir de meilleures différenciations moyennes dans les *Collections* que nous générons avec au moins une de nos méthodes lorsque la taille de la *Collection* est réduite. La section suivante présente une approche à nouveau fondée sur les graphes, permettant d'identifier de petites *Collections* très différenciées.

5.2.1 Identifier de petites Collections en se basant sur les graphes

D'abord, on a fait l'hypothèse que parmi les *Collections* générées avec nos algorithmes, on peut identifier de *sous-collections* de *Tests* fortement différenciés deux à deux. Et pour vérifier cela, on va utiliser les graphes à nouveau. Alors, pour avoir une petite *Collection de Tests* différenciés, nous commençons par la génération d'une assez grande *Collection*. Ensuite, nous identifions des sous-ensembles de *Tests* fortement différenciés deux à deux

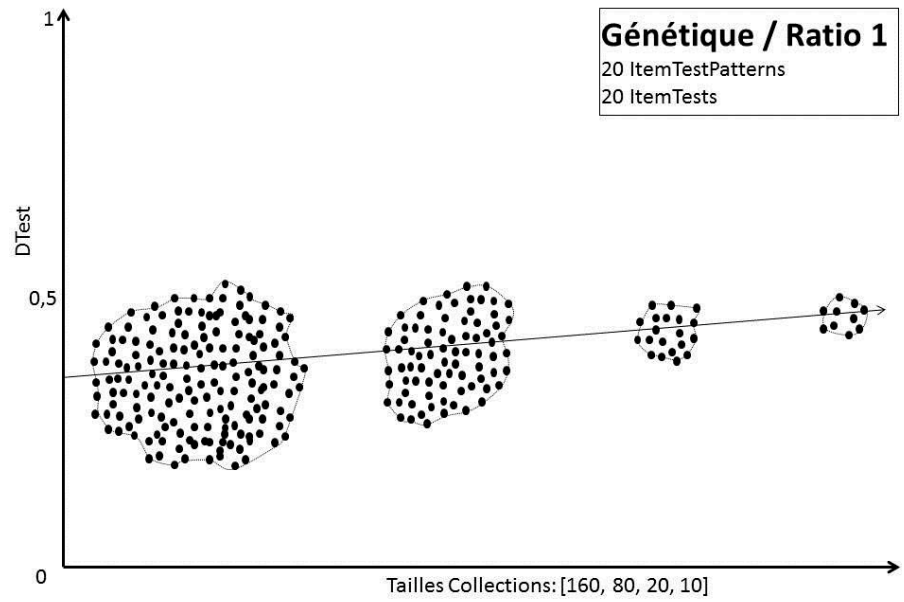

FIGURE 5.5 – La distance moyenne pour des tailles de collection différentes avec l'algorithme génétique

au sein de plus grandes *Collections*. Conceptuellement, nous représentons les *Tests* et leurs distances à l'aide de graphes. Ainsi, une *Collection* est représentée par un graphe $G_T = (S, A)$ (Graphe des *Tests* distants). Chaque sommet du graphe représente un *Test*, et il y a une arête entre deux sommets si la distance entre les deux *Tests* est supérieure ou égale à un seuil. Ainsi, S est l'ensemble des *Tests* allant de T_1 à T_n et A={$\exists(T_i, T_j)$} ssi T_i et T_j sont assez distants.

En effet, le fait de chercher un sous ensemble de *Tests* fortement distants deux à deux au sein d'une *Collection*, si on place un seuil minimal de distance qui caractérise une arête, il revient à identifier un sous graphe complet dans un graphe, c'est-à-dire une clique. Une clique est un graphe dont tous les sommets sont adjacents deux à deux [9]. Supposant qu'on souhaite utiliser 4 *Tests* pour une évaluation dans un salle dont la topologie se représente comme le graphe planaire 5.2. La figure 5.6 présente un graphe G_T (*Collection* de neuf *Tests* où chaque arête signifie que les deux extrémités sont distantes au-delà d'un seuil s prédéfini par l'enseignant) où les sommets $t1$, $t2$, $t3$ et $t4$ constituent une clique de *Tests* distants. C'est-à-dire que ces 4 *Tests* ont tous une distance deux à deux supérieur à un seuil prédéfini et constituent par conséquent une petite *Collection* très différenciée pour une distribution dans 5.2.

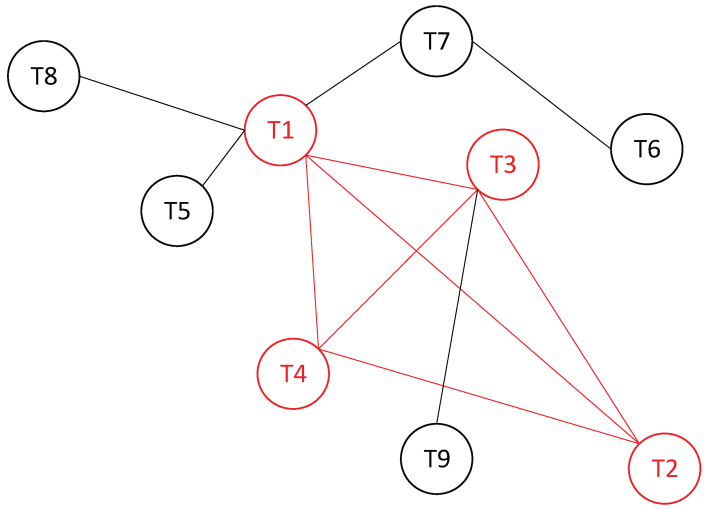

FIGURE 5.6 – Exemple de clique de quatre Tests dans G_T, un graphe de 9 sommets

Alors, si on considère nos différentes méthode de génération de *Tests* différenciés, à quel niveau on peut procurer de bonnes différenciations dans de petites *Collections* ? La section suivante présente une série d'expérimentations qui ont été menées en ce sens.

5.3 Protocole d'expérimentation et résultats

Nos expérimentations ont visé à estimer la taille des cliques présentes dans le graphe représentant une *Collection* générée par l'algorithme Aléatoire (la moins performante) et l'algorithme Génétique (la plus performante). L'objectif est de voir avec quelle méthode on arrive à identifier des cliques de *Tests* les plus différenciées possible. Pour cela, dans chaque *Collection* générée, *DTest()* est calculée entre chaque couple de *Tests*, pour être ensuite comparée à des seuils fixés empiriquement à 0.5, 0.6 et 0.7 pour les ratios respectifs 1,1.5 et 2. Empiriquement, nous avons fixé la taille minimale des cliques recherchée à 4, sachant que la topologie des salles de classe sont généralement faite pour que les apprenants s'asseyent en rangée. Les paramètres de génération étant fixés comme suit :

- $X1$ représente la taille de la base d'*ItemTestPatterns*. Cette taille correspond à la quantité d'*ItemTestPatterns* dont dispose l'enseignant dans sa base de données. Ce paramètre varie entre *[20, 30 et 40]*. Dans chaque *ItemTestPatterns*, il y a 6 *RepItemITPs*. Nous avons choisi empiriquement ces trois tailles de base source.

- $X2$ représente la taille de la *Collection* de *Tests* à générer ; cette taille correspond à la quantité de *Tests* que l'enseignant souhaite avoir dans sa *Collection*. Ce paramètre varie sur seulement les valeurs *[4, 6, 8, 10, 15 et 20]*.
- $X3$ représente la taille de chaque *Test*. Cette taille correspond à la quantité d'*ItemTests* que l'enseignant souhaite avoir dans chaque *Test*. Ce paramètre est fixé à *20*. Considérant les valeurs de $X1$, le fait de fixer $X3$ à 20 permet de travailler sur les ratios de 1, 1.5 et 2.
- $X4$ représente la quantité de *RepItemITs* par *ItemTest*. Donc, la quantité de choix pour chaque *ItemTest* qui est fixée à 4.

La suite de cette section présente les premiers résultats obtenus avec cette approche de génération des *Collections* de taille réduite.

5.3.1 Identification des cliques avec les algorithmes aléatoire et génétique

La figure 5.7 résume une partie des résultats pour l'observation de cliques de 4 *Tests* dans les *Patrons de Collections* générés. Alors, pour un ratio de 1, ni la méthode Génétique celle de l'Aléatoire n'ont permis d'obtenir des cliques de taille 4. Pour un ratio de 1.5, des cliques de cette taille sont possibles dès lors que le seuil est inférieur ou égal à 0.6. Pour obtenir des cliques de taille 4, avec un seuil fixé à 0.7, il nous faut utiliser l'algorithme génétique avec un ratio de 2.

THRESHOLD = 0.5		
ALGO / RATIO	RANDOM	GENETIC
1	✗	✗
1.5	✓	✓
2	✓	✓

THRESHOLD = 0.6		
ALGO / RATIO	RANDOM	GENETIC
1	✗	✗
1.5	✗	✓
2	✓	✓

THRESHOLD = 0.7		
ALGO / RATIO	RANDOM	GENETIC
1	✗	✗
1.5	✗	✗
2	✗	✓

FIGURE 5.7 – Cas où les algorithmes aléatoire et génétique génèrent des cliques de 4

Nous voyons plus clairement dans la matrice 5.8 que dans le cas des petites *Collections*, la méthode Génétique offre une meilleure différenciation. Par exemple, dans le cas de 5.8, seulement un couple de *Tests* de toute la *Collection* qu'a généré l'algorithme aléatoire a une distance supérieure ou égale au seuil. Contrairement aux résultats de l'algorithme génétique qui offre 9 couples de *Tests* entre lesquels il existe une distance supérieure ou égale au seuil, ce qui nous donne des cliques de 3 *Tests* tous distants d'au moins 0.6 (deux à deux).

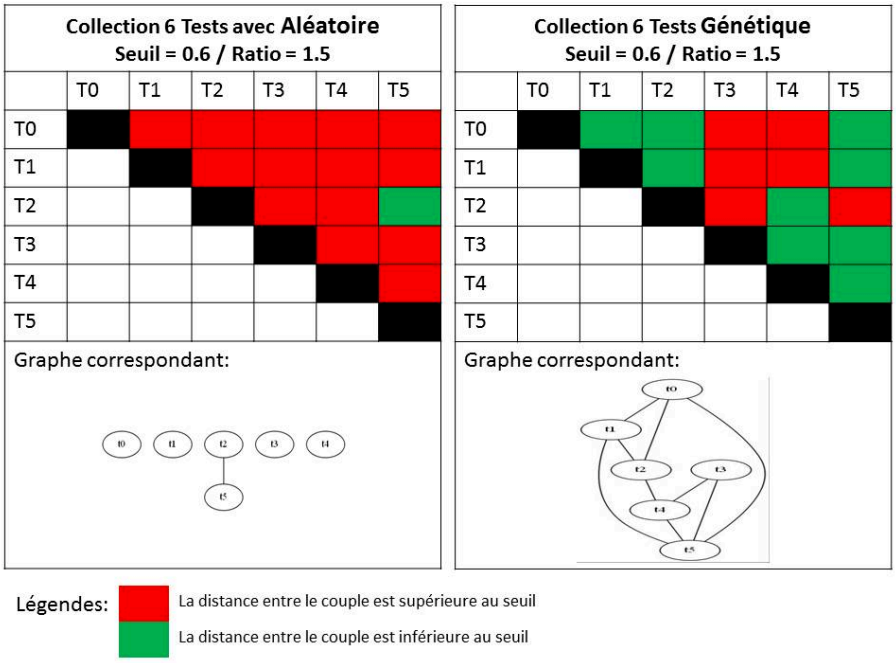

FIGURE 5.8 – Visualisation des *Collections* sous le format des graphes

Dans l'objectif de voir si l'on peut obtenir encore mieux avec l'approche Génétique, les mêmes expérimentations ont été refaites en gardant les mêmes paramètres en faisant passer de 1000 à 2000 la quantité d'itérations et de 3600 secondes à 7200 secondes le TimeOut. En résultat, aucune amélioration n'a été constatée.

5.3.2 L'influence des réponses sur la différentiation dans les petites *Collections*

Les premières expérimentations de la méthode de génération Aléatoire avaient démontré que la quantité de choix liés à un *ItemTestPattern* a la capacité d'impacter la différenciation moyenne dans les *Collections* générées. Cette augmentation de différenciation peut varier de 0.4 à 0.7. Ainsi, dans le cas des petites *Collections* une nouvelle expérimentation a permis de mesurer à quel point les choix pouvaient modifier la différenciation dans les petites *Collections* pour obtenir des cliques de *Tests* encore plus différenciées. En d'autres termes, nous avons cherché à voir à partir de quel ratio de *RepItemITPs/RepItemITs* on génère des cliques de 4 *Tests* encore plus distants.

Le protocole suivi pour cela fut la génération de *Collections* de 4, 6, 8, 10, 15 et 20 *Tests* avec l'algorithme génétique où tous les *Tests* sont faits de 20 *ItemTests* et de 4 *RepItemITs* par *ItemTest*. Les *Collections* sont générées depuis des bases de 20, 30 et 40 *ItemTestPatterns*. On fait varier la quantité de *RepItemITPs* de 6, 7, 8, 9 et 10 pour chaque

vague de génération. Les distances seuils considérées sont 0.5, 0.6 et 0.8. Pour l'algorithme génétique, pour chaque *Collection*, on considère une population de 500 *Collections* au départ, le pourcentage de suppression est fixé à 8%, le pourcentage de mutation est fixé à 3%, la génération s'arrêtera à 2000 itérations ou au timeout de 7200 secondes.

En Résultats, 8 est le minimum de *RepItemITPs* qu'il faut pour avoir des cliques de 4 *Tests* au minimum assez distants dans notre cas de figure (tirage de 4 RepItems). Parce que :

- A partir de 8 *RepItemITPs* on peut générer des cliques de 4 *Tests* pour une *DTest()* seuil de 0.5 à partir du ratio 1.
- A partir de 8 *RepItemITPs* on peut générer des cliques de 8 *Tests* pour une *DTest()* seuil de 0.6 à partir du ratio 1.5.
- A partir de 8 *RepItemITPs* on peut générer des cliques de 3 *Tests* pour une *DTest()* seuil de 0.8 à partir du ratio 2.

En résumé, pour les ratios 1, 1.5 et 2 d'*ItemTestPatterns/ItemTests*, si on a un ratio de *RepItemITPs/RepItemITs* supérieur ou égal à 2, on peut générer des cliques de 4 *Tests* distants de 0.5 pour le ratio 1, des cliques de 8 *Tests* distants de 0.6 pour le ratio 1.5 et des cliques de 3 *Tests* distants de 0.8 pour le ratio 2. Le schéma 5.9 montre un tableau qui récapitule les résultats pour des *Collections* de 6 *Tests*, il s'agit là d'un exemple visuel qui présente en colonne de gauche les Ratios, en colonne de droite les seuils et en horizontal les différentes tailles de *RepItemITP*. La colonne du milieu qui représente la taille 8 témoigne qu'il s'agit là de la quantité minimale de *RepItemITPs* qu'il faut avoir pour obtenir au moins des cliques de 3 pour les ratios 1, 2, 3 avec des seuils 0.5, 0.6 et 0.8.

Grâce à ces premiers résultats, on a pu remarquer qu'avec la méthode de génération génétique, on arrive à avoir une meilleure différenciation moyenne dans les *Collections* de taille réduite comme nous l'avions prédit dès l'hypothèse de départ. Par exemple sur une *Collection* de 300 *Tests*, générés à partir d'un ratio de 1.5 dans les premières expérimentations, on avait obtenu une distance moyenne de 0.58, pourtant cette différenciation moyenne est passée à plus de 0.68 pour une *Collection* de 15 *Tests*, provenant d'un même ratio (avec un ratio de choix égal à 2).

5.4 Conclusion

En conclusion, ce travail permet principalement d'apporter une autre manière d'appréhender la contrainte de la différenciation. Il est à retenir que ce problème a été modélisé sur un format de graphe, ce qui offre une ouverture vers un ensemble de méthodes de résolutions de problèmes s'appuyant sur la théorie des graphes, et qui permet de formaliser

		ReptITP						
		6	7	8	9	10		
Ratio	1	Moy : 0.4259 M-C : 1 T-E : 573.13 s Schéma :	Moy : 0.4707 M-C : 2 T-E : 558.11 s Schéma :	Moy : 0.5020 M-C : 3 T-E : 547.69 s Schéma :	Moy : 0.5244 M-C : 6 T-E : 535.85 s Schéma :	Moy : 0.5464 M-C : 6 T-E : 541.83 s Schéma :	0.5	Seuil
	1.5	Moy : 0.6113 M-C : 3 T-E : 330.08 Schéma :	Moy : 0.6436 M-C : 5 T-E : 425.37 s Schéma :	Moy : 0.6694 M-C : 6 T-E : 419.53 s Schéma :	Moy : 0.6840 M-C : 6 T-E : 415.23 s Schéma :	Moy : 0.6963 M-C : 6 T-E : 415.30 s Schéma :	0.6	
	2	Moy : 0.7215 M-C : 2 T-E : 367.93 s Schéma :	Moy : 0.7438 M-C : 2 T-E : 366.31 s Schéma :	Moy : 0.7589 M-C : 2 T-E : 359.65 s Schéma :	Moy : 0.7745 M-C : 2 T-E :358.83 s Schéma :	Moy : 0.7870 M-C : 2 T-E : 357.58 s Schéma :	0.8	

FIGURE 5.9 – Cas où l'algorithme aléatoire et génétique génèrent des cliques de 4

encore mieux le problème de la différenciation. Plus spécifiquement nous représentons l'espace d'épreuve d'évaluation (physique ou réel) comme un graphe de voisinage et l'affectation des *Tests* consiste en une fonction de coloration de graphe. Nos expérimentations ont montré que cette méthode permet d'augmenter la différenciation, sauf que cela ramène le questionnement de l'attribution des *Tests* en salle de classe de façon à ce que les étudiants voisins reçoivent toujours des *Tests* différenciés.

Il sera très certainement question de continuer à creuser cette piste pour vérifier si on peut trouver d'autres méthodes de génération encore plus performantes qui pourraient permettre de maximiser encore plus la différenciation dans nos *Collections*. Notamment, nous n'avons pas encore vérifié la performance des méthodes de génération par la Sélection sur les petites *Collections*. Il faudra aussi étudier dans quelle mesure il est possible d'être plus souple sur les seuils comme dans le cas où il manquerait une ou deux arêtes pour constituer une clique distante, c'est-à-dire accepter d'ajouter ces dernières pour un seuil légèrement inférieur à celui qui était prédéfini. Il sera également question de donner suite à notre approche d'attribution des *Tests* par la fonction de coloration de graphes et également de voir s'il existe d'autres approches similaires à tester.

Le chapitre suivant fait une conclusion générale de ces travaux de thèse en commençant par un bilan pour finir sur les grandes perspectives de DIFAIRT.

Chapitre 6

Conclusion générale

Ce manuscrit a présenté les travaux de recherche qui ont été menés dans le cadre de cette thèse. L'objectif principal fut de réfléchir sur la question de la génération des sujets d'évaluation différenciés dans le contexte des universités. La différenciation est pratiquée dans différents contextes et/ou pour différentes raisons dans l'enseignement supérieur comme expliqué dans le premier chapitre. La question principale qu'a posé cette thèse fut comment maximiser la différenciation moyenne dans des *Collections de Tests* auto-générées, tout en minimisant la taille de la base de questions sources nécessaires. Ainsi, nous nous sommes principalement intéressés aux sujets de types QCM pour les évaluations sommatives/certificatives.

D'un point de vue pédagogique, ce travail constitue un apport dans l'enseignement supérieur. Car, de plus en plus les enseignants et les apprenants utilisent divers outils d'assistance à certaines de leurs taches. On peut parler de l'enseignant qui en utilise pour la préparation de ses cours, pour faire passer ses examens, pour corriger ses copies, etc. Ainsi, l'outil d'aide aux évaluations qui résulte de cette thèse contribue à son niveau à assister la communauté des enseignants dans la gestion de leurs évaluations. D'un point de vue scientifique, cette thèse contribue principalement au domaine des Environnements Informatiques pour l'Apprentissage Humain car les travaux qui existent dans le champs des évaluations certificatives sont encore assez peu comparativement aux autres types d'évaluation telle que l'évaluation formative. De plus, en se basant sur nos recherches bibliographiques, le questionnement de la différenciation dans les *Tests* d'évaluation passés en salle de classe en contexte universitaire n'est pas courant dans la recherche en EIAH. Donc, ces travaux permettent d'ouvrir un nouveau champ de questionnement. Ces recherches ont également contribuées de façon un peu plus secondaire au domaine de recherche des méta-heuristiques, par l'application des algorithmes génétiques moins présent dans la littérature.

Les sections suivantes font respectivement état d'un bilan des travaux présentés dans ce manuscrit et introduit la problématique de l'équité dans les *Collections* générées, qui constitue une de nos perspectives majeures.

6.1 Bilan

Cette thèse a été menée dans le cadre d'un projet du laboratoire MIS de l'Université de Picardie Jules Verne appelé DIFAIRT. Ce projet s'attaque aux différentes problématiques liées à l'évaluation en contexte universitaire. Plus précisément, il s'intéresse à l'automatisation de la conception, de la génération, de la correction et à l'analyse de résultats des *Tests* d'évaluation. Toutefois, cette thèse a été principalement portée sur la partie de la génération de *Tests* différenciés, ce qui a permis de mettre en place le générateur de *Tests* différenciés appelé DIFAIRT-G (DIFAIRT - Generator).

Pour répondre à la question de l'optimisation de la différenciation, une métrique originale appelée DTest() a été élaborée. Elle permet de mesurer le niveau de différenciation entre deux *Tests* dites différenciés en structures et/ou en contenus. Elle a été développée et expérimentée sur des *Tests* de type QCM. Grâce à cette métrique, les performances de trois méthodes de génération de *Collections* de *Tests* différenciés ont été mesurées selon le niveau de différenciation moyenne qu'elles peuvent procurer dans les *Tests* qu'elles gérèrent. La première méthode, basée sur un algorithme de génération Aléatoire procure une différenciation moyenne supérieur à 0.8 dès que l'enseignant dispose de trois fois plus de contenus dans sa base source que dans ses *Tests* (dans le jargon de DIFAIRT, pour un ratio supérieur à 3). La deuxième méthode est construite autour d'un algorithme de Sélection qui a témoigné une meilleure performance par rapport à la méthode Aléatoire, soit 5% plus performante dans les ratios inférieurs à 3. La dernière méthode est construite sur une méta-heuristique (un algorithme génétique) et elle est 10% plus performante que la première méthode. Alors, avec la deuxième et la troisième méthodes de génération, il est déjà possible pour le générateur DIFAIRT-G de générer des *Collections* de *Tests* assez différenciés pour des ratios inférieurs à trois, ce qui suppose qu'on puisse dire à l'enseignant qu'il n'a pas forcément besoin de trois fois plus de contenus dans sa base de données que dans ses *Tests*. Toutefois on s'est demandé si on ne pouvait pas faire mieux en terme de différenciation, et c'est là qu'on change de paradigme.

Ainsi, nous sommes partis sur une nouvelle hypothèse qui suppose qu'un enseignant n'aurait pas forcément ou pas toujours besoin d'un grande *Collection* de *Tests* différenciés tous deux à deux dans le cas d'une épreuve qui aurait lieu dans une salle de classe. En effet, une *Collection* de taille réduite dans laquelle les *Tests* seraient très différenciés deux à deux pourrait être plus profitable (pour limiter les fraudes par exemple) qu'une grande

Collection avec une distance moyennement plus faible. C'est à dire, chaque étudiant n'aura pas forcément un sujet d'évaluation unique, mais seulement les étudiants voisins auraient des sujets différenciés. Nous nous sommes alors basés sur de nouvelles conditions expérimentales pour générer des *Collections* de tailles réduites, tout en pensant à une méthode pour optimiser leur distribution en salle de classe afin que chaque apprenant puisse recevoir un *Test* très différencié de ceux de ses plus proches voisins. Concernant l'attribution des *Tests* en salle d'examen, des pistes de méthodologies d'aide à la distribution optimisée, basées sur l'algorithmique des graphes sont initiées.

A présent, nous sommes capables de proposer aux enseignants une assistance dans la génération de *Tests* différenciés par 3 principales méthodes, basées sur 3 algorithmes distincts. Grâce à la métrique proposée, nous sommes en mesure de contrôler la différenciation au moment de la génération des *Tests*, ce qui permet de l'optimiser, et donc faire en sorte que l'enseignant n'ait plus forcément besoin d'une grande proportion de questions sources pour avoir une différenciation moyennement élevée dans ses *Tests*. En terme d'attribution des *Tests* en salle, nous ne sommes pas loin de pouvoir assister l'enseignant dans une distribution optimisée pour éviter les fraudes.

Pour l'instant, ces premiers travaux de recherche se réunissent au travers d'un prototype en cours de développement. Une des premières tâches pour le court terme est de finaliser ce prototype. Pour l'instant, il y a un prototype fonctionnel car tout le projet est bâti sur l'outil YMCQ qui est toujours en utilisation. Ce prototype est actuellement accessible en mode ligne de commande. Toutefois, ces travaux de thèse ont permis d'initier une nouvelle architecture pour un nouveau prototype de DIFAIRT. Ce prototype sera utilisable depuis une interface graphique nommée AMI-DIFAIRT, ce qui facilitera la manipulation des différentes fonctionnalités et algorithmes du prototype par les enseignants. Il sera alors question de synchroniser AMI-DIFAIRT avec les différents algorithmes de génération développés. A terme, ce prototype sera soumis à une expérimentation qui est son utilisation réelle par de groupes d'enseignants de l'Université de Picardie Jules Verne et de l'École Supérieure d'Infotronique d'Haiti en vue de recueillir leurs critiques sur les performances globales de DIFAIRT.

En ce qui concerne la différenciation dans les *Collections* de taille réduite et la méthode d'optimisation de l'attribution des *Tests*, une étude de faisabilité a été faite grâce aux premières expérimentations. Toutefois, il reste à approfondir cette piste notamment en étudiant de nouvelles topologies d'espace d'examen comme des salles qui se rapprocherait plus d'une topologie de graphes en étoile, en cercle ou encore quelconques. De ce fait, il faudra voir si la coloration de graphe peut toujours aider à optimiser l'attribution des *Tests* différenciés dans ces dernières. Aussi, il sera question d'intégrer cette approche

d'attribution dans le prototype DIFAIRT sous forme d'une fonctionnalité à part et entière.

Cette thèse a surtout offert certaines ouvertures sur différents autres domaines comme : le CSP, car le fait d'avoir pu utiliser un algorithme génétique dans le contexte d'un travail sur les EIAH suppose qu'on peut puiser encore plus loin dans les CSP d'autres approches algorithmiques permettant de résoudre d'autres problématiques en EIAH. Notamment, toutes nos méthodes de générations permettent de créer les *Tests* de façon aléatoire, même si ces derniers sont acceptés ou rejetés par rapport à des conditions pré-établies pour favoriser la différenciation. Ainsi, on se pose la question de s'il existe des CSP qui permettraient de faire de la génération de manière non complètement aléatoire, c'est-à-dire contrôler la différenciation au moment même du tirage des questions et des choix de réponses ; cette thèse offre également des ouvertures sur l'utilisation du formalisme des graphes qui a rejoint cette thèse dans un contexte très spécifique qui est l'utilisation de la coloration de graphe pour optimiser l'attribution des *Tests* en salle d'examen. L'avantage de ce coté est que tous les problèmes qu'on peut modéliser sous forme de graphes peuvent profiter des centaines de méthodes de résolutions de problèmes de ce champ.

A moyen terme, nous avons deux perspectives majeures qui sont : premièrement l'intégration de nouveaux types de questions dans DIFAIRT car pour le moment nous ne travaillons qu'avec des QCM ; deuxièmement, l'intégration de la contrainte de l'équité dans les *Collections* que génère DIFAIRT, par le contrôle de la difficulté des *Tests*. La prochaine section introduit les premiers travaux/réflexions qui ont été menés autour de la problématique de l'équité dans les *Tests* différenciés.

6.2 Problématique de l'équité

Des *Tests* différenciés distribués à un ensemble d'apprenants ayant suivi un même cours peut faire l'objet de questionnements au niveau de l'équité entre ces derniers. Comment s'assurer que chaque apprenant soit évalué avec un même niveau d'exigence si les contenus des *Tests* sont différents ? Dans le contexte de nos travaux, l'équité se définit par le fait que les *Tests* d'une même *Collection* soient de niveaux de difficulté équivalents. DIFAIRT avait adopté l'approche la plus simple qui consiste à allouer des poids de difficulté aux questions de la base source. Nous envisageons ici une méthode plus élaborée pour permettre d'améliorer la qualification de difficultés des questions sources en fonction de la façon dont elles ont été répondues dans les épreuves antérieures. Cette section présente un état de l'art sur l'équité et la perception de cette contrainte dans DIFAIRT.

6.2.1 État de l'art

Le Learning Object Metadata (LOM) a été publié par l'Institut d'Électronique et des Ingénieurs en Électronique. Ce standard permet de présenter les méta-données pour les ressources liées à l'apprentissage en ligne et non numérique. Il a été reconnu comme norme en 2002. Ces méta-données sont présentées sous 9 catégories regroupant 68 éléments. Parmi les neuf sections, la section "Educationnal" contient les caractéristiques pédagogiques de toutes ressources d'enseignement. Ce dernier comporte onze attributs, dont un attribut nommé Difficulty. Cet attribut présente cinq choix (niveaux) qui sont les suivants : très facile, facile, moyen, difficile, très difficile [55]. D'après LOM, un Item peut donc avoir cinq niveaux de difficulté. Cependant, il est difficile de nous appuyer uniquement sur cette approche étant donnée qu'un *ItemTest* est construit à partir d'un *ItemTestPattern* et que l'enseignant peut tout au plus tenter de définir un niveau intrinsèque pour l'*ItemTestPattern* mais pas pour un *ItemTest*.

IMS-GLC (Système de gestion pédagogique - Global Learning Consortium) est reconnu pour avoir développé de nombreuses normes dans le domaine de l'éducation. Notamment, comme cela a été présenté dans le chapitre 1, QTI (Interopérabilité des questions et des *Tests*) est une norme de IMS-GLC qui est dédiée aux évaluations. Dans sa version 2.0, elle adopte les neuf sections de LOM (sauf certains attributs comme la difficulté), en ajoutant une dixième section, nommée qtiMetadata, dans l'objectif de remédier à certains manques de ressources pour l'évaluation. Dans le document officiel où QTI présente ces méta-données, nous pouvons lire le message suivant : *"Note that the LOM-defined Interactivity Type, Interactivity Level, Semantic Density, Intended End User Role, Typical Age Range and Difficulty fields are not recommended by this profile."* ce qui signifie que les types comme l'interactivité, le niveau d'interactivité, la densité sémantique, le rôle de l'utilisateur final prévu, la tranche d'âge typique et les champs de difficulté définis par LOM ne sont pas recommandés [62]. Ceci voudrait dire que la difficulté comme méta-données n'a pas grande importance pour QTI. Ainsi, il nous semble difficile de nous appuyer sur les normes de caractérisation des objets d'apprentissages pour nous aider à caractériser la difficulté de nos *Tests* et *ItemTests*.

Dans le laboratoire d'informatique de l'Université Pierre et Marie Curie (LIP6), l'équipe MOCAH a présenté une chaîne éditoriale de schémas d'exercices mathématiques permettant aux enseignants de mieux former et évaluer leurs étudiants. Il s'agit d'évaluations non sommatives. Un même programme d'exercices peut générer des exercices avec différents niveaux de difficulté. La solution selon eux consiste donc à imposer des contraintes aux paramètres des modèles d'exercice en vue de gérer la difficulté. Comme la chaîne éditoriale est basée sur QTI pour assurer l'interopérabilité des modèles, ils ont proposé une extension de QTI en ajoutant la classe TemplateConstraint à la classe abstraite TemplateRule

de QTI [5]. La gestion d'équité en terme de difficulté, serait alors considérée dans le cas de MOCAH comme un ensemble d'exercices créés à partir d'un même schéma respectant les mêmes contraintes. Si c'est le cas, cela revient au fait que l'enseignant concepteur de la base des exercices seul peut définir la difficulté des Items. Toutefois, ce travail nous inspire dans le sens où l'on peut également proposer une extension de QTI qui nous permettrait de gérer la difficulté dans le cas précis qu'est le nôtre.

La section suivante présente des premières approches de DIFAIRT sur la contrainte de l'équité.

6.2.2 Equité d'après DIFAIRT

L'objectif premier de DIFAIRT est d'être capable de générer des *Tests* d'évaluation différenciés. Mais, il lui tient aussi de pouvoir distribuer aux apprenants un ensemble de *Tests* d'évaluations équitables. Comme pour les autres travaux et normes sus-cités, l'équité signifie pour DIFAIRT un ensemble de *Tests*, avec un même niveau de difficulté ou un niveau de difficulté sensiblement équivalent. Lorsque des *Tests* différenciés sont préparés manuellement par un enseignant, ce dernier prend soin de bien choisir les contenus afin de maintenir cette équité entre eux. Mais est-il vraiment en mesure d'assurer que tous les étudiants seront évalués à un même niveau d'exigence ? Lorsqu'il passe par un logiciel comme DIFAIRT il peut porter une attention particulière à la difficulté des *Tests* auto-générés. En premier lieu, au moment de la conception de la base source, l'enseignant a la possibilité de définir un niveau de difficulté par défaut pour chaque *ItemTestPattern*. Il s'agit alors d'une estimation de l'enseignant du niveau de difficulté pour l'*ItemTestPattern* par extension pour tous les *ItemTests* qui pourraient être générés à partir de lui. Cependant, cette approche est arbitraire puisque la difficulté est supposée par rapport à un public et de plus, le tirage des choix de réponses a un grand impact sur la difficulté de l'*ItemTest*. Une approche complémentaire consiste alors à s'appuyer sur les retours statistiques des épreuves d'évaluation pour préciser la difficulté des *ItemTestPatterns*. Ces deux niveaux d'attention sont présentés dans les deux prochaines sous-sections.

L'équité par la probabilité de bien répondre

Le projet DIFAIRT permet à l'enseignant concepteur de la base source d'attribuer une valeur à chaque élément ajouté, quantifiant son niveau de difficulté. Cette information sur la difficulté est considérée comme une information additionnelle sur l'*ItemTestPattern*. Cependant, elle n'est pas vraiment prise en compte au moment de la génération. L'échelle de difficulté de DIFAIRT varie sur l'intervalle [-2, 2] et ces valeurs traduisent la probabilité qu'un étudiant réponde correctement à la question. En quelque sorte, cet intervalle peut se définir empiriquement ainsi :

- -2 : signifie que la probabilité de répondre correctement est très faible, donc de 0 à 20%
- -1 : signifie que la probabilité de répondre correctement est faible, donc de 20 à 40%
- 0 : signifie que la probabilité de répondre correctement est moyen, donc comprise entre 40% et 60%
- 1 : signifie que la probabilité de répondre correctement est forte, donc de 60 à 80%
- 2 : signifie que la probabilité de répondre correctement est très forte, donc de 80 à 100%

Ainsi, la question la plus facile dans la base prend la valeur maximale de l'échelle qui est 2, ce qui signifie que l'étudiant a une probabilité de plus de 80% de répondre correctement à la question. A l'inverse, la question la plus difficile à être répondue correctement prend la valeur minimale de l'échelle qui est de -2, ce qui veut dire que l'étudiant à moins de 20% de chance de bien répondre à la question. Pour les autres cas de figures, la question est plus ou moins difficile si son indice de difficulté est plus ou moins proche de 2 ou de -2. Il est important de noter que l'*ItemTestPattern* qui est saisie par l'enseignant sans aucun indice de difficulté prend par défaut l'indice 0, ce qui veut que la question à une probabilité de 50% d'être répondue correctement. Par contre il est à noter que le fait de disposer d'une valeur de difficulté pour un *ItemTestPattern* ne veut pas forcement dire qu'on connaît la difficulté d'un *ItemTest* généré à partir de ce dernier. Si on prend l'exemple d'un *ItemTestPattern* qui contient 12 choix de réponses, il est possible de générer des *ItemTests* différents avec des jeux de choix distincts. Alors, certains jeux de choix peuvent rendre la question plus favorable à être bien répondue que d'autres (ne serait-ce que par élimination), même s'ils proviennent du même *ItemTestPattern*. Toutefois, ces poids de difficulté intrinsèques sont utilisés comme difficultés par défaut, mais sont appelés à s'affiner au fur et à mesure des épreuves d'évaluation.

L'équité par les statistiques

Le deuxième niveau d'attention qu'apporte le projet DIFAIRT à la contrainte d'équité commence après la composition des étudiants. Plus spécifiquement, elle se base sur la phase des analyses statistiques et sémantiques des résultats après chaque épreuve d'évaluation. L'approche est de se dire qu'il est bien de partir sur une difficulté arbitraire renseignée par l'enseignant au moment de la création de la base, mais qu'il est encore mieux de pouvoir confirmer, d'infirmer et/ou de réajuster cette difficulté en fonction des évaluations passées.

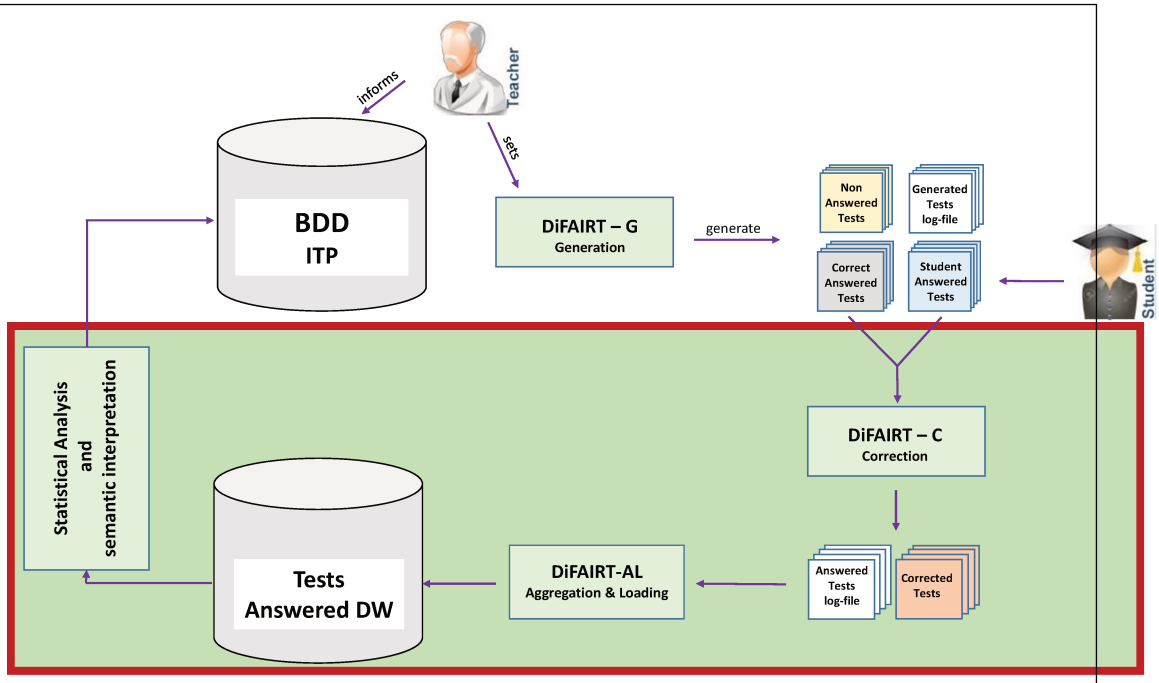

FIGURE 6.1 – La partie concernant la sauvegarde des résultats et leur analyse statistique et sémantique

De premiers travaux [1] ont permis de mettre en place une structure qui correspond à cette nouvelle approche. Pour cela, le cycle de fonctionnement 2.1 de DIFAIRT a été enrichi. Dans la figure 6.1, la partie encadrée présente les éléments modifiés du cycle initial. Plus concrètement, les changements sont les suivants :

- le système de correction automatique de DIFAIRT a été baptisé DIFAIRT-C (DIFAIRT - Corrector) ;
- le processus de correction donne naissance à deux principaux fichiers qui sont les *CorrectedTests* qui sont les copies corrigées et les *AnsweredTestLog-File* qui contient l'historique de correction, c'est-à-dire toutes les réponses et le fait qu'elles aient été cochées ou non ;
- la base *TestResults* est remplacée par un entrepôt de données (Data Ware house) nommé *TestAnsweredDW*, ce qui permet d'agréger les données pertinentes issues des épreuves pour une analyse statistique ;
- les données de l'entrepôt sont agrégées et chargées grâce à un nouveau composant nommé *DIFAIRT-AL* (Agregation and Loading) ;
- et l'étape finale est toujours le composant d'analyse statistique et sémantique des données de l'entrepôt pour un enrichissement de la base de questions sources.

1. Les pistes de solution proposée dans cette partie ont été majoritairement menées par Simon Caillard, stagiaire de Master 2 en 2017, sous l'encadrement de l'équipe de DIFAIRT.

L'objectif de cette évolution du projet DIFAIRT est de proposer une structure efficace et pertinente pour permettre le recueil de données sur les *Tests* corrigés. Puis en s'appuyant sur cette structure, de proposer un système qui permet d'obtenir des indicateurs nous renseignant sur la difficulté des *ItemTests* auxquels les étudiants ont répondus, pour ainsi en déduire des informations sur les *ItemTestPatterns* desquels les *ItemTests* ont été générés. Après certaines analyses, ces indicateurs viendront alors enrichir la base des *ItemTestPatterns*. Une fois la gestion de l'équité intégrée dans nos algorithmes de génération, on sera en mesure de proposer des *Tests* de plus en plus équitables d'épreuves en épreuve. Maintenant, la question devient, quels indicateurs peuvent être pertinents et comment les calculer à partir des données recueillies ? La section qui suit présente les premières propositions de calcul de la difficulté dans DIFAIRT.

6.3 Quelques propositions pour l'équité dans DIFAIRT

Une épreuve de composition est faite grâce à une *Collection* de *Tests*, où chaque *Test* contient alors un ensemble d'*ItemTests* et que ces derniers sont composés d'un *EnonceItemIT* et d'un nombre de *RePItemITs*. Pour cela, nous sommes partis de l'hypothèse que la difficulté d'un *Test* varie en fonction des *ItemTests* qui le composent et de leur placement. Mais que, la difficulté de l'*ItemTest* peut varier en raison du jeu de *RepItemITs* qu'il contient.

6.3.1 Indicateurs et entrepôt de données

Le principal défi technique est que la structure de stockage soit évolutive en fonction du besoin d'analyse souhaité. Aussi, qu'elle soit capable de conserver au fil du temps et de manière exhaustive, les informations relatives aux *ItemTests* répondus, provenant de l'ensemble des *Tests* générés. Elle doit faciliter la gestion/manipulation de grands volumes de données car si on considère l'exemple d'une épreuve pour 100 d'étudiants, avec 100 *Tests* différenciés qui contient 40 *ItemTests* chacun, il existe alors potentiellement 4000 *ItemTests* différents et donc également, 16000 *RepItemITs* qui ont été cochés ou non.

De ce fait, le choix technique pour de premières expérimentations est porté sur les entrepôts de données du modèle en étoile, couramment utilisé, qui permet de gérer de grande quantité d'informations historisées et non volatiles. Ce modèle consiste en une table de faits, avec autour d'elle, un ensemble d'autres tables. Ces dernières contiennent les éléments descriptifs du fait et sont nommées, dimensions. La table de faits contient alors les données observables, celles que l'on possède sur le sujet que l'on souhaite étudier, selon divers axes d'analyses (dimensions) [36]. Dans l'entrepôt de données *TestAnsweredDW* qui a été modélisé, on y retrouve la principale table de faits : *IT-Checking-Test* qui regroupe les informations liées au plus bas niveau de granularité, à savoir le choix de

réponse (*RepItemIT*) coché ou non. Les tables de faits *IT-Checking-Exam* et *IT-Results-Test* quant à elles, permettent de stocker les données concernant les *ItemTests* répondus et les résultats obtenus par les étudiants ; et pour finir la table *IT-Results-Exam*. Elles permettent d'aborder les informations stockées suivant différents degrés d'analyse. Ces informations sont regroupées au niveau d'une Épreuve donnée et concernent les *ItemTests* ainsi que les *RepItemITs* générés au cours de ce dernier. Plus concrètement, en observant les tables *IT-Checking-Exam* et *IT-Results-Exam*, les informations disponibles sont les suivantes :

- Le nombre de fois qu'un *RepItemIT* donné, d'un *ItemTest* donné, dans un contexte donné a été coché et qu'il devait l'être pour avoir juste.
- Le nombre de fois qu'un *RepItemIT* donné, d'un *ItemTest* donné, dans un contexte donné n'a pas été coché alors qu'il aurait dû l'être.
- Le nombre de fois qu'un *RepItemIT* donné, d'un *ItemTest* donné, dans un contexte donné a été coché et qu'il ne devait pas l'être.
- Le nombre de fois qu'un *RepItemIT* donné, d'un *ItemTest* donné, dans un contexte donné n'a pas été coché et qu'il ne devait effectivement pas l'être.
- Le nombre de fois qu'un *ItemTest* donné dans un contexte donné aura été répondu juste, en fonction du fait qu'il ait un barème strictement positif ou non.
- Le nombre de fois qu'un *ItemTest* donné dans un contexte donné aura été répondu partiellement-juste, en fonction du fait qu'il ait un barème strictement positif ou non.
- Le nombre de fois qu'un *ItemTest* donné dans un contexte donné aura été répondu faux, en fonction du fait qu'il ait un barème strictement positif ou non.
- Le nombre de fois qu'un *ItemTest* donné dans un contexte donné n'aura pas été répondu, en fonction du fait qu'il ait un barème strictement positif ou non.

Pour déterminer la difficulté d'un *ItemTest*, on propose d'agréger toutes les données issues de différentes épreuves, au niveau d'un enseignement donné, dans une filière donnée et en fonction d'un niveau d'étude donné. Ainsi, deux grands types d'informations sont à charger dans l'entrepôt. Les premiers concernent les *RepItemITs*, une table permet d'engranger au fil du temps les données issues des *RepItemITPs* préalablement cumulées au niveau des épreuves en fonction d'un contexte d'enseignement. Des champs spécifiques vont permettre de faire la somme des occurrences des *RepItemITPs*, mais aussi de connaître le nombre de fois que ces derniers ont été coché ou non, pour chaque *ItemTest* dans lesquels ils sont apparus et donc de manière générale pour l'*ItemTestPattern* duquel découlent ces *ItemTests*. On retrouve principalement, le nombre d'occurrence (tous *Tests* confondues) d'un *ItemTest* donné associé à un *ItemTestPattern* donné, dans un contexte d'enseignement donné, à travers un champ *"NbOfOccurrence"*. Aussi, on garde le nombre de fois qu'il a été coché via le champ *"NbOfTimesChecked"*. Ces données permettent d'obtenir des

informations sur chaque réponse en elle-même, dans un contexte donné.

Le second type d'informations à agréger concerne les *ItemTests*. Une nouvelle table, nommée *IT-Context* est ajoutée à la base source d'*ItemTestPattern* dans but de faire le cumule des informations (tous *Tests* confondus) de chaque *ItemTest* généré. Sachant qu'un *ItemTestPattern* permet de générer un certain nombre d'*ItemTests* différents. Le but est d'enregistrer chaque configuration d'*ItemTests* (donc *ItemTest* étant déjà apparu au cours d'un ou plusieurs *Test*), ainsi que les informations statistiques qui lui sont associées comme :

- *NbOfOccurrence* : qui permet de connaître le nombre de fois que cet *ItemTest* est apparu.
- *NbOfTimesTrue* : qui permet de connaître le nombre de fois que cet *ItemTest* a été répondu de manière correcte par les étudiants.
- *NbOfTimesPartTrue* : qui permet de connaître le nombre de fois que cet *ItemTest* a été répondu de manière partiellement correcte par les étudiants.
- *NbOfTimesFalse* : qui permet de connaître le nombre de fois que cet *ItemTest* a été répondu de manière incorrecte par les étudiants.
- *NbOfTimesAbs* : qui permet de connaître le nombre de fois que cet *ItemTest* n'a pas été répondu par les étudiants.
- *NegScale* : qui permet de savoir si le barème associé à l'*ItemTest* comporte des points négatifs en cas de réponse fausse.
- *NbOfTrueAnsITP* : qui permet de connaître le nombre de *RepItemITPs* de type juste qui constituent cet *ItemTest*.
- *NbOfAnsITP* : qui permet de connaître le nombre de *RepItemITPs* qui constituent cet *ItemTest*.

Toutes ces informations sur les *ItemTests* répondus par les étudiants sont alors liées à l'*ItemTestPattern* duquel elles découlent et permettent ainsi au fur et à mesure des données collectées de préciser la difficulté intrinsèque des *ItemTestPatterns*, mais surtout de calculer la difficulté d'un *ItemTest* au moment de sa création.

6.3.2 Calcul de la difficulté

A partir de l'identifiant de la filière dans la table *ContextTeaching*, de celui de l'*ItemTestPattern* ainsi que de ceux des *RepItemITPs*, *DIFAIRT-G* devrait être en mesure de déterminer la difficulté de chaque *ItemTest* généré. Trois principaux cas de figure sont alors possibles :

Cas1 : Des données sont disponibles en quantité suffisante concernant cet *ItemTest*
Plus le nombre d'occurrences d'un *ItemTest* donné est grand, plus les données disponibles

concernant celui-ci seront fiables. Pour les premières expérimentations, nous avons empiriquement décidé qu'un minimum de 10 occurrences est nécessaire pour obtenir des données fiables. En supposant que ce nombre soit atteint, il est alors possible de calculer différentes probabilités concernant un *ItemTest*. Par exemple, il peut être aisé de définir la probabilité de répondre correctement à un *ItemTest* par le rapport du nombre de fois il a été répondu juste par son nombre d'occurrences. A l'inverse, celle de le répondre faux est le rapport du nombre de fois qu'il a été répondu faux par son nombre d'occurrences.

Cas2 : Des données sont disponibles, concernant cet *ItemTest*, mais en quantité insuffisante Dans ce cas de figure, le nombre d'occurrences de l'*ItemTest*, dont on cherche à déterminer la difficulté, est donc inférieur à 10. Donc, les données sur la difficulté de cet *ItemTest* ne sont pas considérées comme fiables. Pour pallier ce problème, on propose de rechercher des données statistiques concernant les *ItemTests* similaires à ce dernier. Un *ItemTest* de la même filière sera considéré similaire s'il possède au maximum deux *RepItemITs* différents, pour un *ItemTest* ayant plus de 6 *RepItemITs*, et 1 seul *RepItemIT* au maximum de différent pour ceux qui en compte moins de 6. Noter que ces valeurs ont été décidées et déterminées pour ne pas dépasser un seuil de 30% de différence, par rapport à l'*ItemTest* dont on cherche à déterminer la difficulté. Au-delà de ce seuil, nous considérons que les données ne sont plus pertinentes.

Alors, pour calculer la difficulté de l'*ItemTest*, il a été décidé de pondérer les données à disposition (les valeurs 60% et 40% sont ici arbitraires) :

- L'*ItemTest* 100% similaire (donc celui dont on cherche à déterminer la difficulté et qui possède moins de 10 occurrences) compte pour 60%
- La moyenne des *ItemTests* similaires compte pour 40%

Cas3 : Aucune donnée n'est pas disponible concernant cet *ItemTest* mais l'*ItemTestPattern* a déjà été utilisé lors d'épreuves antérieures La difficulté de l'*ItemTest* peut être déterminée de manière relativement précise dans ce cas de figure, en se servant des données statistiques disponibles concernant les *RepItemITS* constitutifs de l'*ItemTest*. Pour rappel, ces données sont disponibles dans une table et permettent de connaître le nombre d'occurrences des *RepItemITs*, le nombre de fois qu'il a été coché et si la *RepItemITs* de type "vrai" ou "faux" dans le contexte utilisé pour la génération des *Tests*. Ces informations sont disponibles par l'utilisation de ces *RepItemITPs* dans d'autres *ItemTests* dans la même filière.

Cas4 : Aucune donnée n'est pas disponible concernant cet *ItemTest* et l'*ItemTestPattern* n'a jamais été utilisé lors d'épreuves antérieures Dans ce cas de figure, c'est la diffi-

culté intrinsèque de l'*ItemTestPattern* qui est utilisée. Il s'agit du poids de difficulté par défaut que l'enseignant avait spécifié lors de la conception de la base source.

6.4 Synthèse

Cette section a permis d'aborder la principale perspective de nos travaux de thèse, à savoir comment prendre en compte la problématique de la difficulté dans les *Collections de Tests* générées. Elle permet d'aborder la difficulté d'une part dès la conception de la base de questions source et d'autre part après les épreuves d'évaluation, grâce à des analyses statistiques des données recueillies sur les résultats des étudiants évalués. Cette approche nécessite d'être complètement mise en place et expérimentée sur différentes épreuves d'évaluation pour mieux analyser sa pertinence. Notamment, le fait de disposer d'une valeur de difficulté pour un *ItemTest* ou un *ItemTestPattern* ne veut pas dire qu'on dispose d'une difficulté globale pour un *Test*. Donc la première question est comment on détermine la difficulté d'un *Test* à partir des données statistiques sur ses éléments constitutifs ? D'un autre coté, disposer d'une valeur de difficulté pour un *Test* ou d'une valeur de difficulté moyenne pour un *Patron* de *Collection* ne permet pas encore de parler d'équité. Car, il sera important de déterminer à quel niveau on accepte qu'il y a équité ? En d'autres mots, en dessous de quel ecartype on dit que la *Collection* de *Tests* est équitable ?

D'autres pistes restent encore à creuser et à formaliser sur cette contrainte d'équité, et le coté technique à peaufiner et à mettre en place. Notamment, sur le moyen terme, il sera question de trouver la façon d'inclure cette contrainte dans les différents algorithmes de génération existants et de l'ajouter comme fonctionnalité dans le prototype de DIFAIRT. Aussi, il conviendra d'explorer des pistes de l'intelligence artificielle, surtout qu'il est assez courant de retrouver des travaux en EIAH qui intègrent des techniques d'IA. Par exemple, on peut penser à l'utilisation des traces de résultats d'examen pour créer un modèle de type régression linéaire qui permettrait de prédire la difficulté d'un *Test* dans une *Collection* au moment de sa génération.

Bibliographie

[1] Catherine Agulhon. « La professionnalisation à l'université, une réponse à la demande sociale ? » In : *Recherche et formation* 54 (2007), p. 11-27.

[2] Valérie Albouy et Thomas Wanecq. « Les inégalités sociales d'accès aux grandes écoles suivi d'un commentaire de Louis-André Vallet ». In : *Économie et statistique* 361.1 (2003), p. 27-52.

[3] Linda Allal. « Vers une pratique de l'évaluation formative ». In : *Brussels : De Boek* (1991).

[4] Melle DEBBAH Amina. « Un Système Bio-Inspiré pour l'Adaptation Pédagogique dans un EIAH ». In : ().

[5] Odette Auzende, Hélène Giroire et Françoise Le Calvez. « Propositions d'extensions à IMS-QTI 2.1 pour l'expression de contraintes sur les variables d'exercices mathématiques ». In : *Environnements Informatiques pour l'Apprentissage Humain (EIAH 2007)*. INRP. 2007, p. 47-58.

[6] Monique Baron. « EIAO, quelques repères ». In : *Terminal* 65 (1994).

[7] Beekast. *Beekast, plateforme interactive intégrant un espace d'échange pour encourager la prise de parole, et des activités pour générer des idées, évaluer et faciliter la prise de décision*. 2019. url : `https://www.beekast.com/fr/` (visité le 17/07/2019).

[8] Paul E Black. « Fisher-yates shuffle ». In : *Dictionary of algorithms and data structures* 19 (2005).

[9] Immanuel M Bomze et al. « The maximum clique problem ». In : *Handbook of combinatorial optimization*. Springer, 1999, p. 1-74.

[10] Eric Bruillard. *Les machines à enseigner*. Hermès, 1997.

[11] Baptiste Cablé, Nathalie Guin et Marie Lefevre. « Un outil auteur pour une génération semi-automatique d'exercices d'auto-évaluation ». In : *6e Conférence sur les Environnements Informatiques pour l'Apprentissage Humain*. 2013, p. 155.

[12] Christophe Choquet et al. « Actes de la 6e Conférence sur les Environnements Informatiques d'Apprentissage Humain (EIAH 2013) ». In : *Actes de la 6e Conférence sur les Environnements Informatiques d'Apprentissage Humain (EIAH 2013)*. Presses de l'IRIT. 2013.

[13] Richardson Ciguene et al. « Automatic generating of differentiated assessment tests within higher educational context ». In : *EdMedia+ Innovate Learning*. Association for the Advancement of Computing in Education (AACE). 2018, p. 370-377.

[14] Matthieu Cisel et Éric Bruillard. « Chronique des MOOC ». In : *Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation* (2013).

[15] IMS QTI Components. *IMS Question and Test Interoperability (QTI) : Assessment, Section and Item Information Model Version 2.2.2*. 2018. URL : https://www.imsglobal.org/question/qtiv2p2p2/QTIv2p2p2-ASI-InformationModelv1p0/imsqtiv2p2p2_asi_v1p0_InfoModelv1p0.html (visité le 20/11/2018).

[16] QCM Direct. *Création et correction automatique de QCM*. 2018. URL : https://exatech-group.com/solutions-numeriques/correction-automatique-des-qcm (visité le 20/11/2018).

[17] Johann Dréo et al. *Métaheuristiques pour l'optimisation difficile*. 2003.

[18] Guillaume Durand. « La scénarisation de l'évaluation des activités pédagogiques utilisant les Environnements Informatiques d'Apprentissage Humain ». Thèse de doct. Université de Savoie, 2006.

[19] éduscol. *Portail, éduscol pour l'école de la confiance*. 2019. URL : http://eduscol.education.fr/pid33060/banqu-outils-pour-l-evaluation.html (visité le 23/01/2019).

[20] AE Eiben et Zs Ruttkay. « Constraint satisfaction problems ». In : (1997).

[21] Laure Endrizzi et Olivier Rey. « L'évaluation au cœur des apprentissages ». In : (2008).

[22] Eval-Box. *L'outil d'automatisation des évaluations QCM, Eval-Box*. 2018. URL : https://evalbox.fr/ (visité le 20/11/2018).

[23] Santosh K Gaikwad, Bharti W Gawali et Pravin Yannawar. « A review on speech recognition technique ». In : *International Journal of Computer Applications* 10.3 (2010), p. 16-24.

[24] Sébastien George et al. *Actes de la 7ème conférence sur les Environnements Informatiques pour l'Apprentissage Humain*. 2015.

[25] IMS GLC. *Instructional Management System – Global Learning Consortium*. 2018. URL : https://www.imsglobal.org/ (visité le 20/11/2018).

[26] David E Goldberg et Kalyanmoy Deb. « A comparative analysis of selection schemes used in genetic algorithms ». In : *Foundations of genetic algorithms*. T. 1. Elsevier, 1991, p. 69-93.

[27] Sabine Graf et Beate List. « An evaluation of open source e-learning platforms stressing adaptation issues ». In : *Advanced Learning Technologies, 2005. ICALT 2005. Fifth IEEE International Conference on*. IEEE. 2005, p. 163-165.

[28] Nathalie Guin et al. *Actes de la Conférence EIAH 2017*. 2017.

[29] Charles Hadji. *L'évaluation, règles du jeu : des intentions aux outils*. Éditions ESF, 1989.

[30] Jean-Paul Haton et Marie-Christine Haton. *L'intelligence artificielle*. Presses universitaires de France, 1989.

[31] Claire Hewson. « Can online course-based assessment methods be fair and equitable ? Relationships between students' preferences and performance within online and offline assessments ». In : *Journal of Computer Assisted Learning* 28.5 (2012), p. 488-498.

[32] Ari Huhta. « 33 Diagnostic and Formative Assessment ». In : *The handbook of educational linguistics* (2008), p. 469.

[33] Stéphanie Jean-Daubias. « De l'intégration de chercheurs, d'experts, d'enseignants et d'apprenants à la conception d'EIAH ». In : *Technologies de l'Information et de la Connaissance dans l'Enseignement Supérieur et de l'Industrie*. Université de Technologie de Compiègne. 2004, p. 290-297.

[34] Tommy R Jensen et Bjarne Toft. *Graph coloring problems*. T. 39. John Wiley & Sons, 2011.

[35] Céline Joiron et al. « Automatiser la génération et la correction d'évaluations individualisées en contexte universitaire présentiel ». In : *6e Conférence sur les Environnements Informatiques pour l'Apprentissage Humain*. 2013, p. 43.

[36] Ralph Kimball et Margy Ross. *The data warehouse toolkit : the complete guide to dimensional modeling*. John Wiley & Sons, 2011.

[37] Vipin Kumar. « Algorithms for constraint-satisfaction problems : A survey ». In : *AI magazine* 13.1 (1992), p. 32.

[38] Larousse. *Définition de : "évaluation" dans le dictionnaire Larousse*. 2019. url : https://www.larousse.fr/dictionnaires/francais/%5C%C3%5C%A9valuation/31794 (visité le 23/01/2019).

[39] Bénédicte Le Grand et al. « Restitution aux enseignants de l'évaluation des apprentissages dans des EIAH ». In : *Atelier Évaluation des Apprentissages et Environnements Informatiques de la conférence EIAH 2015 (Environnements Informatiques pour l'Apprentissage Humain)*, 2015.

[40] Marie Lefevre et al. « ASKER : un outil auteur pour la création d'exercices d'auto-évaluation ». In : *Atelier EAEI (Évaluation des Apprentissages et Environnements Informatiques)-Conférence EIAH 2015*. 2015.

[41] Vladimir I Levenshtein. « Binary codes capable of correcting deletions, insertions, and reversals ». In : *Soviet physics doklady*. T. 10. 8. 1966, p. 707-710.

[42] Apprendre En Ligne. *Positionnement historique du terme EIAH*. 2018. url : http://apprendreenligne.over-blog.com/2016/10/positionnement-historique-du-terme-eiah.html (visité le 20/11/2018).

[43] Linternaute. *Définition de : "évaluation" dans le dictionnaire Linternaute*. 2019. url : https://www.linternaute.fr/dictionnaire/fr/definition/evaluation/ (visité le 23/01/2019).

[44] Bertrand Marne, Benjamin Huynh-Kim-Bang et Jean-Marc Labat. « Articuler motivation et apprentissage grâce aux facettes du jeu sérieux ». In : *EIAH 2011-Conférence sur les Environnements Informatiques pour l'Apprentissage Humain*. Editions de l'UMONS, Mons 2011. 2011, p. 69-80.

[45] Jean Mayer. « Le théorème des quatre couleurs : notice historique et aperçu technique ». In : *Cahiers du séminaire d'histoire des mathématiques* 3 (1982), p. 43-62.

[46] Patrick Mendelsohn et Pierre Dillenbourg. *Le développement de l'enseignement intelligemment assisté par ordinateur*. Presses Universitaires de France, 1993.

[47] Riichiro Mizoguchi et Jacqueline Bourdeau. « Entretien avec Riichiro Mizoguchi : Le rôle de l'ingéniérie ontologique dans le domaine des EIAH ». In : *Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation* 11.1 (2004), p. 231-246.

[48] Marie-Annick Montalan. « Modélisation des compétences pour l'entreprise : conception et validation d'un système d'évaluation formative des compétences en comptabilité financière ». Thèse de doct. Toulouse 1, 1998.

[49] Moodle. *Documentation sur l'espace Devoir étudiant dans Moodle*. 2019. url : https://docs.moodle.org/3x/fr/Devoir (visité le 23/01/2019).

[50] Moodle. *Réalisation d'un QCM dans Moodle*. 2019. url : https://docs.moodle.org/19/fr/Ajouter/modifier_un_test (visité le 23/01/2019).

[51] Académie Nancy-Metz. *Évaluation diagnostique, formative, sommative*. 2018. url : https://www4.ac-nancy-metz.fr/svt/evaluation/divers/index.php?idp=177 (visité le 20/11/2018).

[52] Mohammad Norouzi, David J Fleet et Ruslan R Salakhutdinov. « Hamming distance metric learning ». In : *Advances in neural information processing systems*. 2012, p. 1061-1069.

[53] Laura Pappano. « The Year of the MOOC ». In : *The New York Times* 2.12 (2012), p. 2012.

[54] Georges Pecego. « SYGEP, un Système de Génération d'Énoncés de Problèmes dans des domaines variés ». Thèse de doct. Paris 6, 1998.

[55] Jean-Philippe Pernin. « LOM, SCORM et IMS-Learning Design : ressources, activités et scénarios ». In : *actes du colloque «L'indexation des ressources pédagogiques numériques», Lyon*. T. 16. 2004.

[56] Philippe Perrenoud. « Évaluation formative et évaluation certificative : postures contradictoires ou complémentaires ». In : *Formation professionnelle suisse* 4 (2001), p. 25-28.

[57] Philippe Perrenoud. « Les trois fonctions de l'évaluation dans une scolarité organisée en cycles ». In : *Educateur* 2 (2001), p. 19-25.

[58] Neil Postman. « Teaching as a conserving activity. » In : *Instructor* 89.4 (1979).

[59] ProfWeb. *Formative : un outil d'évaluation en ligne*. 2019. url : http://www.profweb.ca/publications/articles/formative-un-outil-d-evaluation-en-ligne (visité le 23/01/2019).

[60] Eval QCM. *Plateforme qui permet de créer des questionnaires en ligne et d'obtenir des réponses en temps réel*. 2018. url : https://www.evalqcm.fr/ (visité le 20/11/2018).

[61] IMS QTI. *Instructional Management System – Question and Test Interoperability*. 2018. url : https://www.imsglobal.org/question/index.html (visité le 20/11/2018).

[62] IMS QTI. *Note that the LOM-defined 'Interactivity Type', 'Interactivity Level', 'Semantic Density', 'Intended End User Role', 'Typical Age Range' and 'Difficulty' fields are not recommended by this profile.* 2019. URL : http://www.imsglobal.org/question/qtiv2p2/QTIv2p2-Metadata-InfoBindModelv1p0/imsqtiv2p2_metadata_v1p0_InfoBindv1p0.html#RootAttribute_QTIMetadata_composite (visité le 17/07/2019).

[63] Ministère de l'Enseignement supérieur de la Recherche et de l'Innovation. *Organisation licence master doctorat (L.M.D.)* 2019. URL : http://www.enseignementsup-recherche.gouv.fr/cid20190/organisation-licence-master-doctorat.html (visité le 07/01/2019).

[64] Jean-Michel Renders. *Algorithmes génétiques et réseaux de neurones*. Hermès Paris, 1994.

[65] Gauthier Roger-François et al. *L'évaluation des étudiants à l'université : point aveugle ou point d'appui ?* 2017. URL : http://www.education.gouv.fr/cid5592/l-evaluation-des-etudiants-a-l-universite-point-aveugle-ou-point-d-appui.html (visité le 20/11/2018).

[66] Open Class Rooms. *Plateforme d'appentissage en ligne*. 2018. URL : https://openclassrooms.com/fr/ (visité le 20/11/2018).

[67] Marilyne Rosselle. « Etude des objectifs d'une plate-forme de coopération pour les EIAH ». In : *Environnements Informatiques pour l'Apprentissage Humain 2003*. ATIEF ; INRP. 2003, p. 379-390.

[68] Gérard Scallon. *L'évaluation formative des apprentissages*. T. 2. Presses Université Laval, 1988.

[69] Lotfi Sofiane Settouti et al. « Systèmes à base de traces pour l'apprentissage humain ». In : *Communication in the international TICE, Technologies de l'Information et de la Communication dans l'Enseignement Supérieur et l'Entreprise* (2006).

[70] SN Sivanandam et SN Deepa. « Genetic algorithms ». In : *Introduction to genetic algorithms*. Springer, 2008, p. 15-37.

[71] Rok Sosic et Jun Gu. « A polynomial time algorithm for the n-queens problem ». In : *ACM SIGART Bulletin* 1.3 (1990), p. 7-11.

[72] Mandavilli Srinivas et Lalit M Patnaik. « Adaptive probabilities of crossover and mutation in genetic algorithms ». In : *IEEE Transactions on Systems, Man, and Cybernetics* 24.4 (1994), p. 656-667.

[73] Pierre Tchounikine. *Conception des environnements informatiques d'apprentissage : mieux articuler informatique et sciences humaines et sociales*. 2002.

[74] Pierre Tchounikine. « Précis de recherche en ingénierie des EIAH ». In : (2009).

[75] Pierre Tchounikine. « Quelques éléments sur la conception et l'ingénierie des EIAH ». In : *Actes des 2ème assises nationales du GdR I3-Groupe de Recherche Information Interaction Intelligence, décembre 2002*. 2002, 13-pages.

[76] Antoine Ting et al. « A syntactic business form classifier ». In : *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*. T. 1. IEEE. 1995, p. 301-304.

[77] Luengo Vanda. *Introduction eiah, cours Haïti*. 2016. URL : https://fr.slideshare.net/VandaLuengo/introduction-eiah-cours-hati (visité le 20/11/2018).

[78] Li Yujian et Liu Bo. « A normalized Levenshtein distance metric ». In : *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), p. 1091-1095.

Table des figures

1.1 Schéma de QTI représentant la décomposition d'une évaluation 16
1.2 Format des questions sources 17
1.3 Exemple de correction avec YMCQ 19

2.1 Cycle général de fonctionnement du système 29

3.1 Exemple de génération d'un *ItemTest* à partir d'un *ItemTestPattern* . . . 34
3.2 Exemple de la distance Hamming 38
3.3 Simulation distance de Levenshtein sur les chaines $M_1 = "ABA"$ et $M_2 = "CAC"$ 40
3.4 Première analogie entre Levenshtein et *DTest* 40
3.5 Schéma de zonage de DTest() 42
3.6 Exemple d'identifiant d'un *ItemTest* 45
3.7 Validation du cas de faible permutation dans les énoncés et de faible disparité dans les réponses (DTest proche de 0) 46
3.8 Validation du cas de forte permutation dans les énoncés et de faible disparité dans les réponses (DTest proche de 0.5) 47
3.9 Validation du cas de forte disparité dans les réponses (DTest proche de 0.7) 47
3.10 Validation du cas de forte disparité dans les énoncés (DTest proche de 1) 48
3.11 Cycle de génération d'une collection d'une collection de tests 50
3.12 Exemple d'un *patron de Test* de 3 questions 52
3.13 Exemple des distances *DTest()* pour un *Patron de Collection* de 100 *Tests*, issu d'une base de 10 *ItemTestPatterns* et formés de 10 *ItemTests* 53
3.14 Exemple du regroupement des *DTest()* pour les *Patrons de Collections* issus d'une base de 10 *ItemTestPatterns* pour des *Tests* formés de 10 *ItemTests* 54
3.15 Nuage de points des distances moyennes dans les Collections regroupées par ratio allant de 1 à 25 55
3.16 Distances moyenne comparées au ratio 56
3.17 Comparatif des collection de *Tests* de tailles différentes pour un même ratio 57
3.18 Augmentation de la distance moyenne grâce à l'augmentation des choix de réponses 58

4.1 Les distances avec la Sélection HARD 65
4.2 Les distances avec la Sélection SOFT 65
4.3 Les distances moyennes de la sélection face à l'aléatoire (ratio 1-3) . . . 66
4.4 Comparaison des distances entre les couples de *Tests* issues d'une collection de 40 *Tests*, faite l'algorithme Aléatoire et celui de la Sélection 67

4.5 Comparaison des distances entre les couples de *Tests* issues d'une collection de 30 *Tests*, faite l'algorithme Aléatoire et celui de la Sélection 67
4.6 Table représentant les seuils à partir desquels la sélection aboutit toujours 68
4.7 Principe de l'opérateur de suppression (8%) 70
4.8 Principe de l'opérateur du crossover 70
4.9 Distance moyenne de l'algorithme aléatoire par rapport à celle de l'algorithme génétique 73
4.10 Distance moyenne, maximale et minimale de l'algorithme aléatoire par rapport à celle de l'algorithme génétique 74
4.11 Distance moyenne de l'algorithme Aléatoire par rapport à celle de l'algorithme génétique 75
4.12 Comparaison de la différenciation moyenne pour les trois méthodes de génération 76
4.13 Architecture actuelle 79
4.14 Architecture AMI-DIFAIRT 81
4.15 Paramétrage de la génération 82
4.16 Interface d'importation et d'exportation des questions 84
4.17 Interface de visualisation et de modifications 85

5.1 Graphe G_P représentant le voisinage dans un amphithéâtre 89
5.2 Exemple de distribution en suivant l'approche de la coloration de graphe 90
5.3 Exemple de représentation d'une salle de classe sous forme de graphe quelconque 91
5.4 La distance moyenne pour des tailles de collection différentes avec l'aléatoire 92
5.5 La distance moyenne pour des tailles de collection différentes avec l'algorithme génétique 93
5.6 Exemple de clique de quatre Tests dans G_T, un graphe de 9 sommets . . 94
5.7 Cas où les algorithmes aléatoire et génétique génèrent des cliques de 4 . 95
5.8 Visualisation des *Collections* sous le format des graphes 96
5.9 Cas où l'algorithme aléatoire et génétique génèrent des cliques de 4 . . 98

6.1 La partie concernant la sauvegarde des résultats et leur analyse statistique et sémantique 108

Table des matières

Résumé **ix**

Remerciements **xi**

Sommaire **xiii**

Introduction générale **1**

1 Contexte pédagogique **5**
1.1 L'évaluation des apprentissages 5
1.1.1 Enseignement supérieur 6
1.1.2 Typologie des évaluations 7
1.1.3 Évaluer en contexte universitaire 8
1.1.4 La différenciation dans les évaluations 10
1.2 Outils au service de l'évaluation des apprentissages 11
1.2.1 Outils d'évaluation à destination des apprenants 12
1.2.2 Outils d'aide aux évaluations pour les enseignants 13
1.2.3 Outils d'évaluation et interopérabilité 14
1.3 YMCQ, un exemple d'automatisation 15
1.3.1 Conception d'une base source 17
1.3.2 Génération de sujets différenciés 18
1.3.3 Correction automatique 18
1.4 Conclusion 20

2 Contexte scientifique et objectifs **23**
2.1 Les Environnements Informatiques pour l'Apprentissage Humain 23
2.1.1 Historique des EIAH 24
2.1.2 EIAH et évaluations des apprentissages 26
2.2 Problèmes de satisfactions de contraintes 27
2.3 Le projet DIFAIRT 28
2.3.1 Principe de DIFAIRT 28
2.3.2 Certains questionnements relatifs à ces travaux 30
2.4 Problématiques et objectifs de nos travaux 32

3 Une métrique pour la différenciation dans les évaluations **33**
3.1 Structure d'un Test et notation 33
3.2 DTest(), la métrique proposée et complexité 36
3.2.1 Inspirations et principes de la métrique 37

3.2.2 Détermination des constantes de valeur de *DTest()* 45
3.3 La métrique *DTest()* face à la génération aléatoire 48
3.3.1 Composants et complexité de l'algorithme aléatoire 48
3.3.2 Protocole d'expérimentation 51
3.3.3 Résultats pour les deux expérimentations 56
3.4 Synthèse 58

4 Génération de sujets différenciés 61
4.1 Génération par la sélection 61
4.1.1 Algorithmes par Sélection : HARD et SOFT 62
4.1.2 Expérimentations et résultats 63
4.2 Génération construite sur une méta-heuristique 68
4.2.1 Principe de l'algorithme génétique 69
4.2.2 Expérimentations et résultats 72
4.3 Bilan et analyse croisée pour les trois méthodes de génération 74
4.4 Prototypage actuel de DIFAIR-G 78
4.4.1 Architecture actuelle 78
4.4.2 Génération des *patrons de Collections* 80
4.5 Vers une interface AMI-DIFAIRT 80
4.5.1 Architecture nouvelle 81
4.5.2 Paramétrage et génération avec AMI-DIFAIRT 82
4.5.3 Constitution de la base source et navigation 83
4.5.4 La correction automatique dans AMI-DIFAIRT 85
4.6 Questionnements de transition au prochain chapitre 85

5 Vers une optimisation de la différenciation : le cas des petites Collections 87
5.1 Méthodologie d'attribution de sujets 88
5.1.1 Représentation du voisinage 88
5.1.2 Affectation des sujets 89
5.2 Meilleure différenciation dans les plus petites Collections ? 91
5.2.1 Identifier de petites Collections en se basant sur les graphes 92
5.3 Protocole d'expérimentation et résultats 94
5.3.1 Identification des cliques avec les algorithmes aléatoire et génétique 95
5.3.2 L'influence des réponses sur la différentiation dans les petites *Collections* 96
5.4 Conclusion 97

6 Conclusion générale 101
6.1 Bilan 102
6.2 Problématique de l'équité 104
6.2.1 État de l'art 105
6.2.2 Equité d'après DIFAIRT 106
6.3 Quelques propositions pour l'équité dans DIFAIRT 109
6.3.1 Indicateurs et entrepôt de données 109
6.3.2 Calcul de la difficulté 111
6.4 Synthèse 113

Bibliographie 115

Table des figures **121**

Table des matières **123**

Génération automatique de sujets d'évaluation individuels en contexte universitaire

Résumé

Ce travail de thèse s'intéresse à l'évaluation des apprentissages et notamment à la génération automatique de sujets d'évaluations dans les universités. Nous nous appuyons sur une base de questions sources pour créer les questions de sujets grâce à des algorithmes qui sont en mesure de construire des *Tests* d'évaluation différenciés. Ces recherches ont permis d'élaborer une métrique qui mesure cette différenciation et de proposer des algorithmes visant à maximiser la différenciation totale sur des *Collections* de *Tests*, tout en minimisant le nombre de patterns nécessaires. Les performances en moyenne de ces derniers dépendent du nombre de patterns disponibles dans la base source (en regard du nombre d'items souhaités dans les *Tests*), et de la taille des *Collections* générées. On s'est focalisé sur la différenciation possible dans de très petites collections de sujets, et propose des pistes méthodologiques pour optimiser la distribution de ces sujets différenciés à des cohortes d'étudiants en respectant les contraintes de l'enseignant. La suite de ce travail sera de prendre en compte le niveau de difficulté d'un *Test* comme nouvelle contrainte, en s'appuyant en partie sur les données statistiques et sémantiques récoltées après chaque Epreuve. Le but est de pouvoir maximiser la différenciation en gardant l'équité entre les *Tests* d'une *Collection*, pour une distribution optimisée lors des Epreuves.

Mots clés : génération automatique, évaluation sommative, eiah, csp, différenciation

---

Automatic generation of individual assessments in higher education

Abstract

This PhD work focuses on the evaluation of learning and especially the automatic generation of evaluation topics in universities. We rely on a base of source questions to create topic questions through algorithms that are able to construct differentiated assessment *Tests*. This research has made it possible to develop a metric that measures this differentiation and to propose algorithms aimed at maximizing total differentiation on *Tests Collections*, while minimizing the number of necessary patterns. The average performance of the latter depends on the number of patterns available in the source database (compared to the number of items desired in the *Tests*), and the size of the generated *Collections*. We focused on the possible differentiation in very small *Collections* of subjects, and proposes methodological tracks to optimize the distribution of these differentiated subjects to cohorts of students respecting the constraints of the teacher. The rest of this work will eventually take into account the level of difficulty of a *Test* as a new constraint, relying in part on the statistical and semantic data collected after each *Test*. The goal is to be able to maximize the differentiation by keeping the equity between the *Tests* of a Collection, for an optimized distribution during the Events.

Keywords: automatic generation, summative assessment, hle, csp, differentiation

---

MIS

Laboratoire Modélisation, Information, Systèmes – 33 Rue Saint-Leu – 80039 Amiens – Picardie – France