Génération automatique de sujets d'évaluation différenciés : vers une distribution optimisée en salle d'examen

Richardson Ciguene

Génération automatique de sujets d'évaluation différenciés : vers une distribution optimisée en salle d'examen

Richardson Ciguené

Laboratoire MIS, 33 rue St Leu, Amiens, Picardie, France
richardson.ciguene@u-picardie.fr

Résumé. Ce travail de thèse s'intéresse à l'évaluation des apprentissages et notamment à la génération automatique de sujets d'évaluation. En s'appuyant sur une base de questions sources, des algorithmes sont en mesure de construire des sujets d'évaluation différenciés. Les travaux menés ont permis de proposer des algorithmes visant à maximiser la différenciation totale sur des ensembles de sujets (nommés *Collection*), tout en minimisant le nombre de questions sources nécessaires. Les performances moyennes de ces algorithmes dépendent du nombre de questions disponibles dans la base source (en regard du nombre de questions souhaités dans les sujets), et de la taille des *Collections* générées. Cet article se focalise sur la différenciation possible dans de 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 voisinage données par l'enseignant.

Mots-clés : Évaluation, Génération Automatique, Optimisation.

1 Introduction

L'évaluation des apprentissages est un volet de la pédagogie dans lequel les EIAH trouvent leur place à différents niveaux. De nombreux types d'évaluation peuvent être instrumentés par des environnements informatiques qui mesurent qualitativement et/ou quantitativement l'apprentissage par rapport à des attendus pédagogiques [1]. Ils peuvent également assister l'enseignant dans la conception et la création des évaluations [2]. Dans ce contexte, nos recherches s'intéressent à la génération automatique d'évaluations différenciées, et l'une des situations pouvant engendrer cela est le souhait de limiter la fraude lors de compositions en amphithéâtre, ou lors de compositions en ligne par sous-groupes en horaires décalés.

L'objectif de cette thèse est de proposer des algorithmes de génération de sujets d'évaluation maximisant la différenciation entre les sujets, tout en homogénéisant leurs niveaux de difficulté, préservant ainsi l'équité. Le premier prototype se repose sur un outil de gestion des évaluations papier appelé YMCQ [3].

Divers outils permettent de générer des QCM. Certains sont payants comme WebMCQ [4], d'autres sont intégrés dans des dispositifs d'apprentissage en ligne, ou sont

spécialisés dans les enquêtes en ligne comme LimeSurvey [5]. Il existe des solutions couvrant tout le processus d'évaluation comme EvalQCM [6] et QCMDirect [7]. En revanche, peu d'entre eux tiennent compte du niveau de difficulté dans un ensemble de sujets générées. De plus, leur approche de différenciation est au minimum fondée sur un mélange d'items, au mieux fondée sur un tirage au sort de ces items dans une base d'items préalablement élaborée. A notre connaissance, aucun de ces outils ne permet de garantir un degré de différenciation ou à l'optimiser.

Nos travaux ont exploré le potentiel de différenciation de plusieurs algorithmes de génération de *Collections* de sujets différenciés et d'en déterminer les seuils de performance. Pour ce faire, une métrique permettant de mesurer la distance structurelle entre deux sujets a été élaborée. Après un rappel des résultats des premiers algorithmes expérimentés, cet article se focalise sur la problématique de la différenciation lors de la génération de *Collections* de petites tailles. Est-il possible d'avoir un taux de différenciation plus élevé lorsqu'on génère de petites *Collections*, et si oui dans quelles conditions et avec quelles garanties ? Une approche fondée sur les graphes de voisinage est ainsi proposée.

2 Générer des collections de sujets différenciés

Plusieurs algorithmes permettent la génération de sujets (*Tests*) d'évaluation différenciés de type QCM. Pour mesurer la performance des algorithmes, une métrique permettant de mesurer la différenciation dans un ensemble de *Tests* a été élaborée. La section 2.1 présente la manière dont cette différenciation est mesurée.

2.1 Mesurer la différenciation

Un *Test* est constitué d'une série d'éléments nommés des *ItemTests*. Pour générer une *Collection* de *Tests*, tout algorithme part d'une base source composée d'*ItemTestPatterns*. Un *ItemTestPattern* comporte un énoncé (*EnonceItem*) et un ensemble, non limité, de réponses possibles associées à cet énoncé (*RepItemITPs*). Pour générer un *Test* de n *ItemTests*, le générateur sélectionne de façon aléatoire dans la base n *ItemTestPatterns* sources. Ensuite pour construire un $ItemTest_{\mathrm{i}}$ avec k choix de réponses, l'algorithme s'appuie sur un $ItemTestPattern_{\mathrm{j}}$, en extrait d'une part l'$EnonceItem_{\mathrm{j}}$ correspondant, puis tire parmi les $RepItemITP_{\mathrm{j}}$ possibles, un sous-ensemble de k éléments correspondant aux choix qui vont alimenter l'$ItemTest_{\mathrm{i}}$. Ce processus est répété autant de fois qu'il y a de *Tests* dans la *Collection*.

Pour mesurer la différenciation, la métrique *DTest()* est élaborée et permet de calculer la distance structurelle existant entre deux *Tests*. *DTest()* est inspirée de la distance de Levenshtein qui permet de mesurer la distance entre deux chaînes de caractères [8]. *DTest()* se base alors sur deux principales fonctions : une mesure de disparité *(disparity())* entre les énoncés présents dans les *Tests*, mais aussi en cas d'énoncés similaires, la disparité dans les choix de réponses associés; une mesure de permutation (*permutation()*) c'est à dire de la distance entre les places des éléments identiques dans les *Tests* (les énoncés de questions en premier lieu, mais aussi les places des

choix de réponses pour un même énoncé). *DTest()* renvoie la valeur 0 lorsque les *Tests* sont totalement identiques, et la valeur 1 lorsqu'ils sont totalement disparates. Grâce à *DTest()*, les performances de trois algorithmes de génération sont mesurées et présentées dans la sous-section suivante.

2.2 Trois algorithmes de génération de tests

Le premier algorithme expérimenté s'appuie sur le processus de génération aléatoire. Un millier de *Collections* de multiples tailles, s'appuyant sur des bases *d'ItemTestPatterns* de tailles variables, a été généré. Pour chacune, la distance entre tous les couples de *Tests* a été calculée. On observe que dans toutes les *Collections*, la distance moyenne se stabilise assez rapidement. Cette valeur est directement proportionnelle au ratio du nombre d'*ItemTestPatterns* par la quantité d'*ItemTests*. Par exemple, la distance moyenne pour des *Tests* constitués de 30 *ItemTests* construits à partir d'une base de 60 *ItemTestPatterns* est la même que pour des *Tests* de 50 *ItemTests* pour 100 *ItemTestPatterns*. Par ailleurs, dès lors que le ratio est supérieur à 3, la distance moyenne dans les *Collections* générées est supérieure à 0.7 (différenciation élevée). Ainsi, il faut avoir 3 fois plus d'*ItemTestPatterns* dans sa base que d'*ItemTests* dans les *Tests* pour avoir une distance moyenne supérieure à 0.7. Un algorithme par sélection [9] s'appuie sur la génération aléatoire avec comme contrainte que chaque *Test* généré soit accepté ou refusé dépendamment de son niveau de différenciation avec les *Tests* générés avant lui. Un gain en performance en moyenne est de 5%, en revanche le temps augmente de façon significative.

Enfin, un algorithme génétique [10] a été élaboré pour tenter de rehausser à nouveau les performances, sur des ratios inférieurs à 3. Ce dernier génère une population initiale de *Collections* de *Tests*. *DTest()* est alors utilisée comme fonction de fitness (qui permet d'ordonner les éléments de la population), ainsi pour chaque *Collection*, la différenciation moyenne est calculée. Elles sont ensuite ordonnées des moins différenciées aux plus différenciées. Des opérations classiques de suppression, de croisement entre les individus les plus forts et de mutation sont effectuées. Ce processus est répété sur plusieurs itérations ce qui permet au fur et à mesure de converger vers des *Collections* de plus en plus différenciées, pour ne conserver que la plus performante à l'issue du processus. Un gain de performance en moyenne de 10% est observé avec cet algorithme, ce qui est significatif sur des valeurs calculées en moyenne, mais les performances sur des ratios inférieurs à 2 restent faibles. De ce fait, nous sommes partis dans un paradigme différent : ne plus réfléchir en moyenne sur des grandes *Collections*, mais générer de petites *Collections* en nous assurant que chaque couple de *Tests* de la *Collection* a une distance supérieure à un seuil (le plus élevé possible). La section suivante présente notre approche de la différenciation dans les petites *Collections.*

3 Différenciation dans les petites collections

Partant de *Collections* déjà générées, nous identifions des sous-*Collections* de petites tailles avec un fort taux de différenciation et répondant à des critères prédéfinis. La section 3.1 présente l'approche déployée s'appuyant sur la théorie des graphes [11].

3.1 Les graphes comme outils pour détecter

Pour identifier des sous-ensembles de *Tests* fortement différenciés deux à deux au sein de plus grandes *Collections*, nous avons représenté les *Collections* et leurs distances à l'aide de graphes. Ainsi, une *Collection* est représentée par un graphe appelé G_T (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. Une fois le graphe G_T construit, il nous faut alors identifier des sous-graphes complets tels que tous les sommets sont reliés aux autres par une arête. Ce type de sous-graphe est appelé une clique en théorie des graphes [12]. La Fig. 1 présente un graphe G_T (*Collection* de six *Tests* où chaque arête signifie que les deux extrémités sont distantes au-delà d'un seuil s) où les sommets *t1*, *t2*, *t3* et *t4* constituent une clique de *Tests* distants.

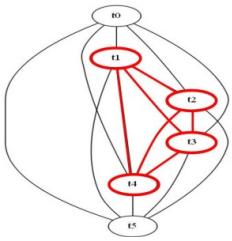

Fig. 1. Exemple de clique de quatre *Tests* dans G_T.

Cette représentation sous forme de graphe permet de ramener ce problème à un domaine très connu, offrant des multiples alternatives de résolution de problèmes complexes. Ainsi, à côté de la différenciation, cela permet d'envisager une intégration plus facile d'autres contraintes telles que la difficulté des *Tests* dans ce même graphe. La section suivante présente les expérimentations menées.

3.2 Premières expérimentations

Nos expérimentations ont permis de vérifier la présence de cliques dans des graphes représentant des *Collections* générées par les algorithmes aléatoire et génétique. Des *Collections* de 4, 6, 8, 10, 15 et 20 *Tests* sont générées. Chaque *Test* est formé de 20 *ItemTests* et de 4 *RepItems*. Les *Collections* sont générées depuis des bases de 20, 30 et 40 ItemTestPatterns (avec 6 *RepItemsITPs*). Pour 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.

Le tableau ci-dessous présente une partie des résultats pour l'observation de cliques de taille 4. Pour le ratio 1, ni le génétique ni l'aléatoire n'ont permis d'identifier des cliques de taille 4. Pour le ratio 1.5, des cliques sont possibles dès lors que le seuil est abaissé à 0.6. Pour obtenir des cliques de taille 4, avec un seuil fixé à 0.7, il nous faut utiliser l'algorithme génétique et être dans 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	✗	✓

Fig. 2. Cas où l'algorithme aléatoire et génétique génèrent des cliques de quatre.

Ces premiers résultats démontrent que nos algorithmes permettent dans certains cas la génération des cliques de quatre *Tests* distants. Il convient alors d'explorer les possibilités de concevoir des algorithmes heuristiques capables de prendre en compte la contrainte de distance entre éléments au moment de la construction des *Tests*. La section 4 présente quelques pistes méthodologiques pour la distribution des sujets.

4 Vers une distribution optimisée en salle d'examen

Supposons que lors d'une épreuve d'examen, le but de l'enseignant soit de limiter la fraude. Dans ce cas, il fait en sorte que les étudiants qu'il estime être voisins obtiennent des *Tests* différenciés. La notion de voisinage est ici relative. Il peut également s'agir d'une contrainte physique imposée par la topologie de la salle d'examen.

Nous représentons une salle de classe à l'aide d'un graphe G_P où chaque sommet représente une place disponible pour un étudiant, et une arête entre deux sommets existe si l'enseignant estime que les deux places sont voisines. La figure suivante illustre un G_P d'un amphithéâtre où l'enseignant considère que chaque place a huit voisins. Déterminer le nombre de sujets nécessaires pour être distribués sur ce graphe et la manière de les répartir sur les sommets renvoie vers le problème de la coloration de graphe [13]. Ce dernier consiste en l'attribution d'une couleur à chaque sommet d'un graphe de sorte que deux sommets reliés par une arête reçoivent des couleurs différentes. Considérant l'exemple de l'amphithéâtre ci-dessous, il apparaît que G_P est un graphe dit 4-coloriable c'est à dire qu'il faut au minimum 4 couleurs pour réaliser sa coloration. Ainsi, si une couleur représente un *Test*, il nous faut 4 *Tests* différenciés pour effectuer une distribution optimale dans un amphithéâtre. Supposons que les couleurs soient nommées A, B, C et D, un exemple de coloration du graphe G_P est présenté dans la Fig. 3.

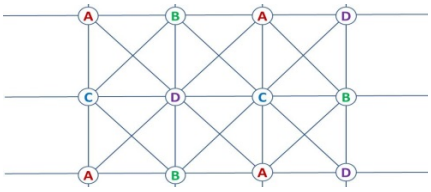

Fig. 3. Exemple G_P pour une salle de type amphithéâtre.

Ainsi, considérant l'exemple de l'amphithéâtre ci-dessus, pour avoir quatre *Tests* assez distants les uns des autres, il faut trouver dans G_T une clique de quatre éléments.

5 Conclusion

Cet article a présenté les avancées d'un travail de thèse qui s'intéresse à la génération automatique de sujets d'évaluation différenciés. Les performances de trois algorithmes de génération sont étudiées en terme de différenciation grâce à la métrique *DTest()*. Toutefois, le paradigme de génération de *collections* de *tests* de petites tailles est expérimenté dans l'objectif d'obtenir encore plus de différenciation dans les *Tests* générés pour une distribution optimisée en salle d'examen. Cette méthodologie est basée sur la théorie des graphes et permet de faire en sorte que les étudiants voisins reçoivent tous des *Tests* différenciés. A court terme, ce travail devra 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*.

References

1. Durand, G.: La scénarisation de l'évaluation des activités pédagogiques utilisant les Environnements Informatiques d'Apprentissage Humain (Doctoral dissertation, Université de Savoie)(2006).
2. Cablé, B., Guin, N., & Lefevre, M.: Un outil auteur pour une génération semi-automatique d'exercices d'auto-évaluation. In 6e Conférence sur les Environnements Informatiques pour l'Apprentissage Humain (p. 155) (2013, May).
3. Joiron, C., Rosselle, M., Le Mahec, G., & Dequen, G.: Automatiser la génération et la correction d'évaluations individualisées en contexte universitaire présentiel. In 6e Conférence sur les Environnements Informatiques pour l'Apprentissage Humain (p. 43) (2013, May).
4. Hewson, C. Web-MCQ, A set of methods and freely available open source code for administering online multiple choice question assessments. Behavior Research Methods 39(3) pp. 471–481 (2007).
5. LimeSurvey Accueil, https://www.limesurvey.org/, dernière connexion 23/01/2018
6. EvalQCM Accueil, http://www.evalqcm.fr/, dernière connexion 23/01/2018
7. QCM Direct Description, https://www.u-picardie.fr/disi/docs/DocQCMDirect/docQCMDirect.pdf, dernière connexion 23/01/2018

8. Levenshtein, Vladimir I.: Binary codes capable of correcting deletions, insertions, and reversals. In : Soviet physics doklady. p. 707-710 (1966).
9. Ciguene, R.: Génération automatique de tests d'évaluation différenciés et equitables. In RJCEIAH (2016).
10. Mitchell, M.: An introduction to genetic algorithms. MIT press (1998).
11. West, D. B.: Introduction to graph theory (Vol. 2). Upper Saddle River: Prentice hall (2001).
12. Bomze, I. M., Budinich, M., Pardalos, P. M., & Pelillo, M.: The maximum clique problem. In Handbook of combinatorial optimization (pp. 1-74). Springer, Boston, MA (1999).
13. Jensen, T. R., & Toft, B.: Graph coloring problems (Vol. 39). John Wiley & Sons (2011)