UNIVERSITÉ DU QUÉBEC À MONTRÉAL

CONCEPTION D'UN ALGORITHME POUR L'ASSIGNATION DE TÂCHES D'ENSEIGNEMENT

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
FRÉDÉRIC THÉRIAULT

AVRIL 2016

REMERCIEMENTS

Je tiens à remercier mon directeur de maîtrise Guy Tremblay, car il a été un mentor idéal pour moi. Il a su me donner la liberté dont j'avais besoin pour apprécier mon mémoire, tout en me redirigeant vers le droit chemin lorsque je faisais fausse route. Je ne pense pas qu'il m'aurait été possible de trouver un meilleur professeur pour me guider. Merci Éric Labonté de m'avoir parlé en si bons termes de ton directeur de maîtrise, tu avais raison sur toute la ligne.

Merci également à tous les professeurs de l'UQAM avec qui j'ai suivi des cours tout au long de la maîtrise. Ils sont tous différents, mais ont la même passion pour leur domaine et c'est avec joie que j'ai pu approfondir mes connaissances dans différentes branches de l'informatique. D'une façon ou d'une autre, ces notions apprises me serviront dans ma carrière.

Ce mémoire n'aurait pas eu lieu sans mon ami et collègue André Cyr, un passionné de la recherche en intelligence artificielle et robotique. D'abord client pour mon entreprise, devenu ensuite aussi un ami, c'est en grande partie grâce à lui si je me suis inscrit à la maîtrise. Ce chercheur dynamique et peu conventionnel est un partenaire idéal et j'espère que nous pourrons continuer à collaborer pendant encore de nombreuses années !

Merci à Arthur et Roxane, mes deux enfants, qui ont si souvent vu leur père se retirer dans son bureau pour faire ses devoirs et travailler sur «l'ordi». Sans comprendre exactement ce que je faisais, ils m'ont aidé à leur façon. Merci également à mes parents, à mes beaux-parents et à ma soeur Sophie pour leur soutien constant et leurs encouragements.

Finalement, un merci sans borne à ma conjointe Laurence, qui m'a soutenu pendant ces trois années d'études. Sa compréhension face à mes absences lors de mes cours et de mes travaux, sa foi inébranlable en ma capacité à réussir et le temps qu'elle a pris à lire et à commenter ce mémoire ont été d'une aide précieuse et irremplaçable. Laurence, ma muse, je t'aime profondément.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

Le problème de l'assignation des tâches d'enseignement du Département d'informatique de l'UQAM est un travail complexe à réaliser étant donné les règles départementales à respecter ainsi que le nombre important de préférences de cours fournies par les professeurs. Afin de simplifier le travail que doit faire la personne responsable de l'assignation des tâches d'enseignement avant chaque session d'enseignement, ce mémoire introduit un algorithme garantissant une optimalité de la solution trouvée. Celui-ci, basé sur différentes techniques de simplification du problème et d'optimisation du temps de recherche, a pour objectif de maximiser les préférences des professeurs tout en respectant les règles départementales. L'analyse des résultats démontre que la plupart des sessions réalistes peuvent être traitées avec cette approche. Afin de bien épauler les différents intervenants qui sont impliqués dans l'assignation des tâches d'enseignement, un logiciel offrant toutes les fonctionnalités nécessaires relatives à ce contexte a également été implémenté. Ce dernier permet également l'utilisation du recuit simulé pour les sessions où l'algorithme introduit dans ce mémoire est incapable d'obtenir un résultat dans un temps raisonnable.

MOTS-CLÉS : assignation de tâches, satisfaction de contraintes, algorithmes complets et approximatifs

INTRODUCTION

Au Département d'informatique de l'UQAM, lorsqu'il incombe au comité exécutif ou à un responsable d'assigner les tâches d'enseignement entre l'ensemble des professeurs, trouver le scénario idéal représente un défi. En effet, il faut tenir compte des règles du département, tout en essayant de distribuer les cours selon les préférences de chacun. Malgré cette complexité avec laquelle il doit composer, le comité doit néanmoins essayer d'optimiser l'assignation des tâches afin de satisfaire le plus de contraintes possible.

Ayant comme objectif le développement d'un outil qui permet de trouver des propositions optimisées à l'assignation des tâches d'enseignement, un étudiant s'est, par le passé, penché sur cette problématique dans le cadre de sa maîtrise [23]. Celui-ci a décrit le problème comme étant de la famille des *constraint-satisfaction-problems* (CSPs) [4]. Il a donc implémenté certains algorithmes applicables à ce type de problème. Malheureusement, l'outil développé dans le cadre de cette recherche n'a jamais atteint un niveau de maturité suffisant pour être utilisé par le département. En effet, selon Monsieur Guy Tremblay (ex-directeur du Département d'informatique de l'UQAM), trois lacunes importantes ont été constatées lors de son utilisation. La première est au niveau des fonctionnalités de l'application qui a été créée. Celle-ci n'a pas été développée comme un logiciel de haut niveau pouvant être utilisé facilement par un utilisateur. La deuxième est que la structure interne de l'application ne facilite pas sa maintenance, ce qui nuit au développement de nouvelles fonctionnalités. La dernière lacune relevée se situe au niveau de la qualité moyenne des résultats obtenus et de la faible consistance

des tests. Bref, l'ensemble de ces faiblesses ont rendu l'outil peu fiable, rigide et défaillant au niveau de son ergonomie. Il est donc, à toute fin pratique, inutilisable.

En considérant que la problématique n'est, à ce jour, pas résolue de façon satisfaisante et que l'étude effectuée précédemment n'a pu aboutir à la création d'un outil fiable, ce mémoire traitera de la conception d'un algorithme dans le contexte de l'assignation des tâches d'enseignement et de l'implémentation d'un outil suffisamment développé et flexible pour être utilisé par le département.

Le chapitre 1 introduit le problème, le chapitre 2 explique comment le problème peut être abordé, le chapitre 3 se concentre sur la technique développée pour le solutionner, le chapitre 4 fait une synthèse des résultats et, finalement, le chapitre 5 résume les grandes caractéristique du logiciel développé comme outil lors de ce mémoire. Ce travail se termine avec une conclusion, rappelant les faits saillants de la recherche ainsi que quelques pistes pour des travaux futurs.

CHAPITRE I

LE PROBLÈME DE L'ASSIGNATION DES TÂCHES D'ENSEIGNEMENT

Pour toute session d'enseignement du Département d'informatique de l'UQAM, une répartition des cours doit être faite à travers l'ensemble des professeurs et des chargés de cours. Pour ce faire, la première étape consiste à demander aux professeurs le nombre de tâches d'enseignement qu'ils désirent offrir ainsi que leurs préférences de cours. Par la suite, le département doit tenter d'optimiser la distribution des cours en satisfaisant au maximum les professeurs face à leurs préférences et ce, tout en respectant les règles départementales. Finalement, suite à cette répartition, les cours qui n'ont pas été attribués aux professeurs sont transmis pour l'attribution aux chargés de cours.

Le problème d'assignation des tâches d'enseignement n'est pas nouveau. En effet, non seulement il a déjà fait l'objet d'un mémoire [23] il y a quelques années, il est également abordé et cité comme exemple dans la littérature [1][5][13][14][19]. Ce chapitre est consacré à la définition du problème et explique pourquoi il n'est pas trivial à solutionner.

1.1 L'assignation des tâches au Département d'informatique de l'UQAM

Avant chaque session, les tâches d'enseignement pour l'ensemble des professeurs doivent être attribuées par une personne responsable. Pour ce faire, les professeurs doivent d'abord fournir à cette personne le nombre de tâches qu'ils désirent

donner à la session suivante, ainsi que leurs préférences parmi les groupes-cours disponibles. Un groupe-cours correspond à un cours et son horaire. Par exemple, le groupe-cours INF7440 (gr. 40) de la session automne 2015 correspond au cours de conception et analyse des algorithmes qui se donne le jeudi soir de 17h30 à 20h30. Il faut noter que lorsque les professeurs soumettent leurs choix de groupe-cours, ces derniers doivent être ordonnés par ordre de préférence, ce qui donne un poids différent à chacun d'eux. Finalement, dans certains cas particuliers, le professeur peut spécifier s'il désire enseigner l'entièreté du cours, ou seulement une partie. Cette possibilité complexifie le problème, comme il sera possible de le constater au chapitre 3.

Suite à la réception des préférences de groupes-cours, la personne responsable de l'assignation des tâches d'enseignement doit tenter de trouver la meilleure assignation possible de groupes-cours aux professeurs. En effet, suite à plusieurs tentatives et ajustements, il devra trouver le scénario qui maximise la satisfaction des professeurs et qui est en accord avec les règles départementales (voir tableau 1.1 et appendice A). Évidemment, il ne doit pas y avoir de conflit d'horaire, puisqu'un professeur ne peut donner deux cours dans des endroits différents en même temps.

La complexité de trouver de bons scénarios de répartition de tâches est très variable. Cela dépend de plusieurs facteurs, tels que le nombre de groupes-cours disponibles, le nombre de professeurs à traiter, le nombre de tâches souhaitées, le nombre de préférences de groupes-cours obtenues par les professeurs, etc. Par exemple, il y a des sessions où peu de professeurs souhaitent enseigner les mêmes cours, ce qui facilite la recherche étant donné la faible concurrence des professeurs pour chaque groupe-cours. Cependant, pour d'autres sessions, trouver un bon scénario est difficile et il y a tant de possibilités à évaluer qu'il est impossible de déduire à vue d'oeil quel serait le meilleur scénario de répartition des tâches. Le tableau 1.2 sert à donner une idée visuelle du nombre de scénarios possibles

Priorité	Règles
1	Un professeur doit soumettre suffisamment de choix de groupes-cours en fonction du nombre de tâches qu'il désire donner.
2	Le directeur de département a priorité sur son choix de groupe-cours et ce, pour deux trimestres par année.
3	Un professeur ayant de mauvaises évaluations pour un cours donné peut perdre sa priorité sur ce cours pour une durée de six sessions.
4	Dans le cas de cours d'études avancées, une priorité est donnée aux nouveaux professeurs et aux professeurs engagés dans le programme commanditaire principal du cours.
5	Si le professeur soumet un groupe-cours dans ses deux premiers choix et qu'il en est le coordonnateur, il a priorité sur le groupe-cours.
6	Un professeur devrait se voir attribuer l'un de ses deux premiers choix de groupes-cours.
7	Un professeur qui donne un cours pour la première fois a priorité pour redonner ce groupe-cours pour deux autres fois.
8	Un professeur qui se voit attribuer un cours aux études avancées n'a pas de priorité pour d'autres cours aux études avancées.

Tableau 1.1 Résumé des règles d'attribution des tâches d'enseignement du Département d'informatique de l'UQAM.

en fonction des préférences fournies par l'ensemble des professeurs. Évidemment, plusieurs scénarios peuvent être facilement écartés et ne valent pas la peine d'être évalués. Malgré tout, même en tronquant une grande partie des scénarios rapidement, le nombre de possibilités intéressantes devient rapidement énorme. Aussi, comparativement au tableau 1.2 qui s'arrête à soixante préférences, une session réaliste dans le Département d'informatique de l'UQAM possède environ une centaine de préférences de groupes-cours à considérer.

Nb de préférences	**Nb de scénarios à évaluer**
10	2^{10}
20	2^{20}
30	2^{30}
40	2^{40}
50	2^{50}
60	2^{60}
...	...

Tableau 1.2 Nombre de scénarios possibles en fonction du nombre de préférences.

C'est dans ce contexte que s'inscrit ce mémoire où il est question d'élaborer une technique pour trouver le scénario optimal d'assignation des tâches d'enseignement, c'est-à-dire celui qui maximise l'attribution des groupes-cours selon les préférences données par les professeurs, tout en considérant et en respectant les règles établies par le Département d'informatique de l'UQAM. Il est à noter que ce travail ne consiste pas à faire l'analyse ou la critique des règles départementales, mais uniquement à trouver une solution qui maximise les assignations tout en respectant les règles, qui sont du ressort de l'assemblée départementale.

CHAPITRE II

LA CATÉGORISATION DU PROBLÈME ET LES AVENUES EXPLORÉES PAR D'AUTRES CHERCHEURS

Afin de bien comprendre comment il est possible de solutionner le problème tel que défini dans le chapitre précédent, il faut tout d'abord introduire certains concepts pertinents et observer comment d'autres études sont parvenues à le résoudre dans leur contexte. Les sections suivantes font un survol de ces notions pertinentes et permettent de mieux cerner l'approche utilisée pour solutionner ce problème.

2.1 Les CSPs

Les problèmes à satisfaction de contraintes (CSPs), ou *constraint satisfaction problems* en anglais, représentent les problèmes qu'il est possible de résoudre en se basant sur trois composantes fondamentales [4] :

1. Un ensemble de variables.
2. Un domaine par variable.
3. Un ensemble de contraintes à respecter.

Un des problèmes les plus représentatifs des CSPs est le problème des n-reines (*n-queen*), où il s'agit de placer les reines de telle sorte qu'aucune d'entre elles n'est

en situation d'attaque avec une autre [1]. Par exemple, dans un scénario à quatre reines, il y aurait quatre variables. Les valeurs que peuvent prendre ces variables, qui représentent les positions possibles d'une reine, correspondent aux cases d'une colonne — une colonne différente par reine —. Ceci correspond au domaine des variables. La contrainte, dans cet exemple, est le fait qu'une reine ne peut être en situation d'attaque avec une autre reine — donc ni sur une même ligne, colonne ou diagonale. La figure 2.1 est un exemple de solution pour le problème des n-reines avec quatre reines à placer.

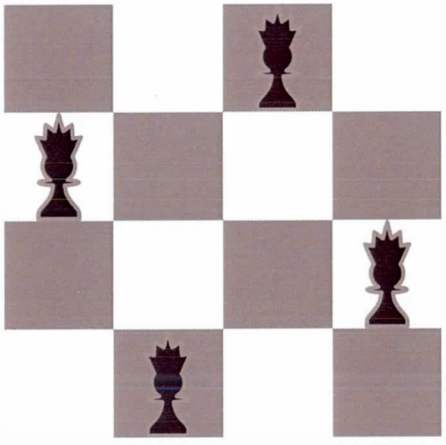


Figure 2.1 Solution du problème des n-reines avec 4 reines.

Plusieurs études portent sur la résolution des CSPs et détaillent différentes techniques qui peuvent être utilisées pour les traiter. La catégorisation de ces dernières diverge dans la littérature. Cependant, de façon générale, on peut compter trois types de techniques. Les sous-sections suivantes représentent un sommaire de ces techniques, telles que présentées dans le livre de Russel et Norvig [19].

---

1. Exemple : `http://frederictheriault.com/experiments/01-csp`

2.1.1 Le *backtracking*

La première technique de résolution de CSPs, le *backtracking*, est aussi décrite dans la littérature comme étant une recherche dans un arbre. Celle-ci consiste à effectuer une recherche en profondeur dans un arbre d'espace d'états afin de trouver une solution au problème.

Le *backtracking* le plus simple est peu performant, puisqu'il s'exécute en parcourant l'arbre de recherche jusqu'à ce que la solution soit trouvée ou que toutes les possibilités (ou branches) aient été explorées. Cependant, il existe plusieurs raffinements qui permettent de converger vers une solution plus rapidement. Par exemple, il est possible de diriger l'exploration de l'arbre de possibilités en choisissant intelligemment le prochain noeud à parcourir. En effet, au lieu de choisir la prochaine variable à assigner arbitrairement, il est possible d'utiliser une heuristique telle que, par exemple, choisir de traiter le noeud ayant le plus petit domaine (*minimum remaining values*) [19]. Bacchus et Run se sont penchés sur l'ordonnancement dynamique de variables dans le but d'accélérer le traitement des CSPs et sont parvenus à avoir un gain de performance de cette façon [2].

2.1.2 La propagation de contraintes

La propagation de contraintes est un type de technique de résolution de CSPs qui est plus facile à comprendre en prenant comme exemple le jeu Sudoku. Ce dernier contient 81 cases (voir figure 2.2), et le but du jeu consiste à réussir à remplir les cases avec des chiffres de un à neuf tout en respectant trois règles :

1. Un chiffre ne peut se retrouver plus d'une fois dans chacun des neuf carrés 3x3 de la grille du jeu.
2. Un chiffre ne peut se retrouver plus d'une fois sur une même ligne de la grille du jeu.

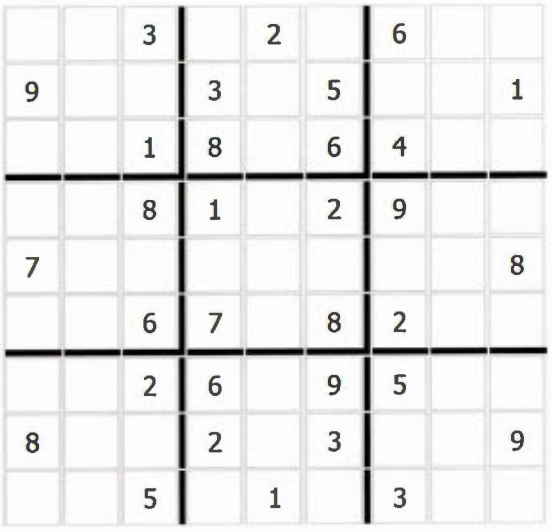


Figure 2.2 Exemple de sudoku (tiré du livre de Russell et Norvig [19]).

3. Un chiffre ne peut se retrouver plus d'une fois sur une même colonne de la grille du jeu.

Pour modéliser le jeu comme un CSP, il est possible de considérer les contraintes comme étant l'ensemble des trois règles, les variables comme étant les 81 cases du jeu et leur domaine, les chiffres de un à neuf.

Dans ce type de problème, il est souvent possible de trouver une solution ou de s'en approcher en traitant le problème par ses contraintes, plutôt que par ses variables. En effet, il s'agit de réduire le domaine des variables jusqu'à ce qu'il soit possible de déduire la bonne valeur à assigner pour chaque variable. L'algorithme AC-3 [18] est un algorithme réduisant progressivement le domaine des variables en éliminant les valeurs impossibles. Par exemple, si une case du jeu possède le chiffre trois, alors il est possible de retirer ce chiffre du domaine des variables de la même ligne et de la même colonne. En recommençant ce procédé pour toutes les variables et aussi longtemps que les domaines peuvent être réduits, il est possible

de solutionner certains sudokus [2]. Cet exemple démontre bien le fonctionnement de ce type d'algorithmes, c'est-à-dire ceux s'attardant aux contraintes plutôt qu'en explorant un espace d'états (ou de possibilités).

2.1.3 Les algorithmes de recherche locale

La dernière approche de résolution de CSPs rassemble les algorithmes de recherche locale. Celle-ci attaque le problème dans son ensemble, c'est à dire que toutes les affectations sont faites aux variables dès le départ, même si certaines d'entre elles contreviennent aux contraintes. Par la suite, des ajustements sont faits sur ces affectations, ce qui permet de converger vers une solution valide.

Parmi les algorithmes de cette catégorie, les métaheuristiques sont efficaces pour résoudre un grand nombre de CSPs. Voici les grandes lignes qui caractérisent un algorithme comme étant une métaheuristique [3] :

1. Il possède des stratégies qui permettent de guider la recherche.
2. Il doit explorer efficacement l'espace de recherche afin de trouver une solution quasi-optimale.
3. Ses techniques utilisées peuvent être de complexité variable.
4. Il est habituellement stochastique (l'exploration est aléatoire).
5. Il peut posséder des techniques permettant de sortir d'optimums locaux — de solutions intéressantes, mais pas optimales.
6. Il n'est pas spécifique à un problème, mais l'heuristique utilisée peut l'être.

Certains exemples de métaheuristiques sont le recuit simulé [21], les algorithmes génétiques [11][24] et la recherche tabou [8][9][10].

---

2. Exemple : `http://frederictheriault.com/experiments/05-sudoku`

2.2 Les COPs

Parmi les CSPs, il existe une branche de problèmes dit d'optimisation, ou *constraint optimization problems* (COPs) en anglais. Celle-ci fait référence aux problèmes où la solution optimale n'est possiblement pas parfaite, c'est-à-dire qu'elle contrevient à certaines règles. Dans ce cas, il s'agit de retourner la meilleure solution possible. Puisque certaines règles peuvent être enfreintes, il faut introduire une nuance : celles qui peuvent être enfreintes (*soft constraints*) et celles qui ne le peuvent pas (*hard contraints*). Cette distinction est importante puisque, dans le cas du problème de l'assignation des tâches d'enseignement, il pourrait être possible qu'un professeur ne donne aucun cours parmi ses deux premières préférences (*soft contraint*), mais il est impossible pour un professeur d'enseigner en même temps deux groupes différents (*hard contraint*). Étant donné que le problème étudié dans le cadre de ce mémoire possède un ensemble d'états fini et qu'il n'existe pas nécessairement de solution qui satisfait toutes les règles départementales selon les préférences reçues des professeurs, il peut être perçu comme un problème faisant partie des COPs.

Pour solutionner les problèmes d'optimisation, il existe deux avenues possibles : l'utilisation d'algorithmes complets ou d'algorithmes approximatifs [3]. Dans plusieurs cas, on y retrouve les mêmes algorithmes que ceux des CSPs, mais appliqués dans un contexte de maximisation des objectifs visés, ou inversement, dans un contexte de minimisation des infractions aux contraintes.

2.2.1 Les algorithmes complets

Les algorithmes complets permettent de trouver la solution optimale à un problème et sont habituellement ceux à considérer en premier lieu. Cependant, ils sont souvent trop lents pour traiter les problèmes dont l'espace de recherche est trop grand à explorer et difficile à réduire. Un des algorithmes de cette catégorie est *branch and bound*, qui est régulièrement utilisé pour résoudre des problèmes d'optimisation [6]. Plus de détails sur cet algorithme sont donnés dans le chapitre 3.

2.2.2 Les algorithmes approximatifs

Étant donné la lacune des algorithmes complets pour les problèmes complexes, les algorithmes approximatifs représentent une alternative intéressante pour les résoudre. Ceux-ci parviennent rapidement à de bonnes solutions, mais ne peuvent toutefois garantir leur optimalité. Les métaheuristiques, telles qu'introduites à la section 2.1.3, sont régulièrement utilisées comme algorithmes approximatifs pour résoudre les COPs.

2.3 Les avenues explorées par d'autres chercheurs

Plusieurs recherches portant sur le problème de l'assignation des tâches d'enseignement par préférences ont été faites. Puisque l'espace de recherche devient rapidement trop vaste pour être exploré par des algorithmes complets, la plupart de ces études se concentrent sur l'utilisation d'algorithmes approximatifs. Par exemple, Xia [23] s'est basé sur l'utilisation d'algorithmes locaux pour tenter de le résoudre. Plus précisément, il a utilisé un algorithme basé sur *min-conflict* [22][19] et le recuit simulé. Une autre étude a tenté de résoudre le même problème pour l'université de Kuwait [1] en utilisant la programmation linéaire. Une autre recherche aborde le problème en utilisant également le recuit simulé, mais combiné au *integer programming* [13]. Plus tard, Gunawan et Ming Ng utilisent une combinaison du recuit simulé et de la recherche tabou pour parvenir à trouver leur solution [12]. Dans le cas de l'étude de Carrasco et Pao, l'utilisation d'un algorithme génétique a été préconisé [5]. Finalement, Hosny se base sur l'escalade (*Hill-Climbing*) pour explorer l'espace de recherche [15]. Après analyse, toutes ces approches sont efficaces et sont en mesure de trouver de bonnes solutions, mais elles ne peuvent toutefois en garantir l'optimalité.

D'un autre côté, Hmer et Mouhoub [14] sont parvenus à utiliser un algorithme complet par l'utilisation de *backtracking* mais sur un nombre restreint de cours et

de tâches à assigner. En effet, les tests ont été évalués sur 17 cours, 10 professeurs avec chacun une tâche à assigner et 4 préférences de cours par professeur. Malheureusement, ces auteurs n'explorent pas le rendement de leur algorithme sur un espace de recherche plus vaste, ce qui limite la portée de l'étude.

Puisque chaque département possède ses propres particularités (règles départementales, nombre de professeurs, etc.), il convient de se demander s'il faut nécessairement se résoudre à utiliser un algorithme approximatif, tel que fait dans la majorité des études précédentes, ou s'il est possible d'utiliser un algorithme complet permettant de trouver le scénario optimal de répartition de tâches, tout en considérant les particularités du Département d'informatique de l'UQAM. C'est ce que nous explorerons dans la suite de ce mémoire.

CHAPITRE III

L'ALGORITHME RFPT ET SA MISE EN OEUVRE

Plusieurs étapes doivent être respectées afin d'être en mesure de fournir une solution adéquate au problème de l'assignation des tâches d'enseignement. Celles-ci constituent l'algorithme RFPT, qui signifie : réduction, fragmentation, parallélisation et traitement. Les sections suivantes expliquent comment le problème a été approché afin de concevoir cet algorithme.

3.1 La représentation

La première étape vers la résolution du problème consiste à lui trouver une représentation adéquate. Tel qu'expliqué précédemment, celui-ci fait partie de la famille des CSPs — plus précisément, des COPs. Il faut donc identifier puis intégrer ses contraintes, ses variables et leur domaine afin d'être en mesure de le résoudre par un ou des algorithmes propres aux CSPs. Les sous-sections suivantes détaillent comment ces éléments ont été représentés.

3.1.1 Les contraintes

Dans le contexte de l'assignation des tâches de cours tel qu'expliqué dans ce mémoire, les règles du département représentent les contraintes du problème. Initialement, il est logique de penser que chaque règle du département possède un poids

(ou un certain nombre de points), et chaque assignation de tâche qui respecte une règle se voit augmenter du poids de cette règle. Ceci en fait donc un problème de maximisation, où la solution retournée est simplement celle qui a le plus grand nombre de points.

Cependant, tel que mentionné dans le précédent mémoire traitant sur le sujet [23] et confirmé par l'ex-directeur du département, Guy Tremblay, les règles sont hiérarchiques. Il faut donc s'assurer d'être en mesure de conserver leur priorité par rapport aux autres. Nous avons conclu qu'un vecteur de points serait plus adéquat, où chaque élément du vecteur représente une règle, de celle qui a le plus d'importance à celle qui en a le moins. La figure 3.1, tirée du logiciel créé lors de la réalisation du présent travail, montre un exemple du vecteur de points accordés pour une assignation fictive.

Figure 3.1 Vecteur de points accordés pour une assignation.

Pour calculer la valeur d'un scénario (d'une solution), il s'agit de faire la somme de tous ses vecteurs de points. Prenons l'exemple d'un scénario à deux assignations. Si la première assignation donne le vecteur de points [1, 0, 0, 0, 0, 14, 0] et la seconde, [1, 2, 0, 0, 0, 13, 0], le scénario peut être évalué en calculant la somme de ces deux vecteurs, soit [2, 2, 0, 0, 0, 27, 0]. Grâce à cette méthode d'évaluation, il est possible de comparer des scénarios entre eux afin de faire ressortir celui qui est préférable, c'est-à-dire celui qui possède le vecteur de points le plus élevé. Par exemple, la figure 3.2 permet de constater que le scénario de droite est préférable

Figure 3.2 Comparaison des vecteurs de points de deux scénarios.

à celui de gauche, puisqu'il possède un plus grand nombre de points à la règle numéro deux.

Toutes les règles du vecteur n'acceptent qu'un pointage positif, à l'exception d'une seule. Il s'agit de la règle du département qui fait référence aux mauvaises évaluations. En effet, si l'assignation d'un groupe-cours à un professeur enfreint cette règle, c'est-à-dire qu'il ne devrait pas donner le cours étant donné ses mauvaises évaluations antérieures, le pointage «-1» lui est attribué. C'est donc une règle où l'on cherche à minimiser le pointage, comparativement aux autres. Afin de la rendre consistante par rapport aux autres règles et de la transformer en un problème de maximisation, le pointage est inversé — il est négatif. Cette technique est typiquement utilisée pour transformer les problèmes de minimisation en problèmes de maximisation.

3.1.2 Les variables et leur domaine

A priori, il est facile de penser que les variables sont les assignations de tâche pour les professeurs et les domaines, quant à eux, sont les préférences de groupe-cours assignables pour chacune de ces variables. Le tableau 3.1 aide à donner une idée de cette représentation. En conceptualisant le problème de la sorte, il s'agirait de faire un tableau, où chaque élément est une assignation de tâche pour un professeur, selon ses préférences (ou «nul» s'il n'est pas possible d'assigner la tâche du professeur).

Variable	prof. #1 (tâche 1)	prof. #2 (tâche 1)	prof. #2 (tâche 2)	...	prof. #n (tâche m)
Domaine	préf. #1, ..., préf. #n, nul	préf. #1, ..., préf. #n, nul	préf. #1, ..., préf. #n, nul	...	préf. #1, ..., préf. #n, nul

Tableau 3.1 Représentation où une variable est une assignation possible de tâche pour un professeur.

Par contre, il ne faut pas oublier qu'il est possible pour le professeur de demander une fraction de tâche (ex : 2.5 tâches voulues). Il se peut aussi que le professeur manifeste sa préférence à donner une fraction d'un groupe-cours en particulier (ex : 50% du groupe-cours INF7215-10). Ces possibilités, quoiqu'intéressantes pour le professeur, peuvent nuire à la performance de l'algorithme. En effet, il faut souligner le fait que, pour tout problème de la famille des CSPs, il faut restreindre autant que possible l'espace de recherche (le nombre de variables et leur domaine). Ceci est dû au fait que le temps requis pour traiter le problème et trouver la solution est directement lié à la taille de l'espace de recherche (le nombre de possibilités à explorer).

Imaginons un professeur qui désire donner 1.5 tâche de cours et fournit cinq préférences de cours à 50% chacune. Cette situation engendre, à elle seule, trois variables dans le tableau d'assignations, où chaque variable correspond à 50% d'une tâche. Les domaines, quant à eux, sont constitués des cinq préférences fournies, en plus de la possibilité «nul». L'espace de recherche pour ce professeur peut donc être calculé par la formule n^t, où n correspond à la taille des domaines et t, le nombre de variables. En utilisant cette formule, il est possible de constater qu'il y a 216 possibilités (6^3) d'assignations à évaluer pour ce professeur.

À l'inverse, en utilisant les préférences comme variables au lieu des domaines, il y a un gain significatif sur la simplicité de représentation du problème et sur la vitesse de traitement. En effet, de cette façon, les domaines possèdent simplement les valeurs «assignée» et «non assignée», tel que le montre le tableau 3.2. L'espace de recherche est donc calculé par la formule 2^p, où p est le nombre de préférences de cours. Pour comparer la situation du paragraphe précédent selon les deux approches, il y a dans ce cas-ci 32 possibilités à évaluer, ce qui est beaucoup moins long à traiter que les 216 possibilités obtenues en utilisant les tâches comme variables !

Variable	préférence #1	préférence #2	...	préférence #n
Domaine	assignée, non-assignée	assignée, non-assignée	...	assignée, non-assignée

Tableau 3.2 Représentation où la variable est la préférence et le domaine est le statut (assigné ou pas).

Cela dit, il existe certains cas où l'utilisation des tâches comme variables est plus avantageuse que l'utilisation des préférences. Ceci est le cas, par exemple, dans une situation où un professeur désire donner une seule tâche et fournit trois préférences. Cependant, puisque la deuxième technique simplifie la représentation du problème en général et offre une plus grande souplesse en ce qui concerne les fractions de tâches et les fractions de préférences, celle-ci semble être une avenue préférable et sera celle utilisée dans ce mémoire.

Lors de la session d'automne 2015, il y a eu 96 préférences de cours fournies par les professeurs. Ceci implique 2^{96} scénarios à évaluer. En se basant sur un test fait avec un ordinateur récent [1], il est actuellement possible d'évaluer environ 10000

1. Test fait avec Google Chrome, sur un ordinateur ayant un processeur quad-core de quatrième génération (i7-4702MQ) et 8 Go DDR3 de mémoire vive.

possibilités par seconde (en considérant les contraintes). À cette vitesse, il faudrait attendre plus qu'une vie avant d'avoir le résultat de la meilleure répartition des tâches à travers l'ensemble des professeurs pour cette seule session.

Dans la prochaine section, nous présenterons une méthode pour accélérer le temps de traitement du problème, puisqu'avec la méthode naïve de base il est impossible d'avoir un résultat dans un temps raisonnable.

3.2 L'algorithme RFPT

Idéalement, tout problème devrait être traité de façon à retourner la solution optimale, c'est-à-dire la meilleure solution possible. Dans le contexte de ce mémoire, celle-ci correspond à la meilleure répartition des tâches aux professeurs, donc celle qui enfreint le plus petit nombre de règles. Puisque le nombre de scénarios possibles à explorer pour chaque session est trop volumineux pour pouvoir les évaluer exhaustivement, il est possible de se tourner vers des algorithmes de recherche approximatif. Ces derniers sont conçus pour trouver une solution aussi bonne que possible étant donné les contraintes de temps, mais pas nécessairement la solution optimale. Cependant, puisque ce type d'algorithme ne garantit l'optimalité de la solution, doit-on nécessairement en utiliser un pour traiter notre problème, ou est-il possible de trouver une autre méthode pour parvenir rapidement à la meilleure solution ?

Pour répondre à cette question, il faut tenter de faire une approximation du temps que nécessiterait un algorithme de recherche complet, c'est-à-dire qui retourne la solution optimale au problème. *Branch-and-bound* [17], un algorithme garantissant la meilleure solution, fonctionne en évitant d'évaluer des branches de possibilités qui ne peuvent pas retourner une meilleure solution que celle actuellement trouvée. Le pseudocode 3.1, tiré du livre de Erciyes [7], correspond aux lignes directrices définissant cet algorithme. Grâce à celui-ci, il est possible, dépendamment du

```
PROCEDURE branchAndBound(T : arbre de possibilités de solutions,
                         v : solution possible ∈ T,
                         b : meilleure solution actuelle)

  SI v est meilleur que b ALORS
    b ← v
  FIN SI

  E ← ensemble des enfants directs de v
  C ← calculerBornes(E)
  C' ← solutionsPrometteuses(C) % c ∈ C tel que c.val + c.valProjetée > b.val

  POUR c ∈ C' FAIRE
    branchAndBound(T, c, b)
  FIN

FIN
```

Pseudocode 3.1 Pseudocode de l'algorithme *Branch-and-bound.*

problème, d'éviter de perdre du temps à évaluer la plupart des possibilités. Pour ce faire, un calcul de bornes permet de constater si la direction prise par l'algorithme vaut la peine d'être évaluée. Dans le cadre de ce mémoire, cela consiste à vérifier, après chaque assignation de variable, si le fait de placer les variables restantes comme étant assignées permet d'avoir une meilleure solution que celle trouvée jusqu'à présent, dans la mesure où ces assignations sont possibles (le groupe-cours est encore disponible et son assignation ne dépasse pas le nombre de tâches désiré par le professeur). Dans la négative, cet espace de recherche peut alors être tronqué puisque même avec un maximum d'assignations restantes, il ne pourra pas retourner une meilleure solution. Ceci dit, même si, par exemple, l'algorithme évitait l'évaluation de la majorité des possibilités, la solution optimale prendrait encore trop de temps à être calculée.

Il est difficile d'accepter de se tourner vers un système qui retourne une bonne répartition des tâches, tout en sachant qu'elle pourrait ne pas être la meilleure. En effet, il est assez gênant de présenter aux professeurs des assignations de tâches sans être certain que ce sont les meilleures assignations possibles. C'est pourquoi, avant d'accepter de se rabattre sur un algorithme de recherche approximatif (qui ne garantit pas la solution optimale), il faut prendre le temps de voir s'il est possible d'appliquer un ou des raffinements au problème, ce qui permettrait de trouver la meilleure solution d'assignations. Les sections suivantes expliquent les raffinements appliqués en vue d'obtenir une solution optimale dans un temps d'exécution court.

3.2.1 Les assignations sans concurrence

L'offre de groupe-cours étant assez grande, certains professeurs donnent comme première préférence un groupe-cours qui n'est voulu par aucun autre professeur. Il est donc possible d'assigner directement ces groupes-cours, puis de les retirer du problème. Cette étape permet de réduire considérablement l'espace de recherche (le nombre de possibilités restantes à évaluer). Dans le cas où la première préférence du professeur lui est assignée, il est possible de refaire le même procédé, puisque sa deuxième préférence pourrait également être sans concurrence. Utiliser cette technique offre un deuxième avantage, puisqu'il est également possible que ce professeur, dans l'éventualité où il ne souhaitait donner qu'une seule tâche, permette à un autre professeur d'avoir l'assignation de sa première préférence. C'est donc un algorithme que l'on recommence tant qu'il y a un professeur qui se voit assigner une préférence sans concurrence. Le pseudocode 3.2 décrit son fonctionnement.

```
FONCTION assignationsSimples( profs : Set{Professeur} ): Set{Assignation}
  assignations ← {}

  REPETER
    aucuneNouvelleAssignation ← VRAI

    POUR prof ∈ profs FAIRE
      SI prof.nbTâchesVoulu > 0 ALORS
        préférence ← prof.préférences.first

        SI préférence est sans concurrence ALORS
          assignations ← assignations ∪ { préférence }
          prof.préférences ← prof.préférences - { préférence }
          prof.nbTâchesVoulu ← prof.nbTâchesVoulu - nbTaches(préférence)
          aucuneNouvelleAssignation ← FAUX
        FIN
      FIN
    FIN

  JUSQUE aucuneNouvelleAssignation

  RETOURNER assignations
FIN
```

Pseudocode 3.2 Pseudocode d'assignation des préférences sans concurrence.

3.2.2 La décomposition du problème en sous-problèmes

C'est en observant attentivement les préférences des professeurs qu'un constat a pu être fait : il existe des groupes de professeurs qui désirent les mêmes cours. Par exemple, certains professeurs aiment donner des cours sur les bases de données, d'autres sur le génie logiciel, et ainsi de suite. Le problème est donc constitué de plusieurs sous-problèmes indépendants, où il s'agit simplement de former des groupes de professeurs qui ont les mêmes préférences de cours et de les traiter indépendamment des autres. Par exemple, s'il y avait eu 3 groupes distincts et de même taille de professeurs souhaitant donner les mêmes cours pour la session d'automne 2015, le problème aurait été grandement simplifié, puisqu'il aurait fallu traiter 3×2^{32} possibilités [2], au lieu de 2^{96}. Ceci représente une réduction exponentielle du nombre de possibilités et permet déjà d'envisager l'utilisation de l'algorithme de recherche *branch-and-bound*. Le pseudocode 3.3 permet, lorsque possible, de fragmenter le problème en plusieurs sous-problèmes. Celui-ci prend en paramètre la liste de groupes-cours à répartir ainsi que la liste de professeurs qui désirent donner au moins une tâche. Son résultat est une liste d'ensembles où chaque ensemble correspond aux groupes-cours dont l'assignation influence les assignations des autres groupes-cours du même ensemble.

L'idée de la fragmentation nous est venue d'une phrase du livre de Russel et Norvig [19], où il est écrit : *"Completely independent subproblems are delicious, then, but rare"*.

Cette optimisation n'est cependant pas toujours possible. En effet, certains professeurs pourraient donner des préférences de cours qui touchent plusieurs groupes, ce qui fait en sorte que deux ou plusieurs sous-problèmes en deviennent un plus gros. Ceci a pour conséquence de reformer un problème trop complexe pour que

---

2. Donc 12884901888 possibilités.

```
FONCTION fragmentation( grpCours: Set{GroupeCours},
                        profs: Set{Professeur} ): Set{Set{GroupeCours}}
  sousProblemes ← {}

  POUR groupeCours ∈ grpCours FAIRE
    ensemble ← trouverEnsembleQuiContient(groupeCours, sousProblemes)

    SI ensemble = {} ALORS
      ensemble ← { { groupeCours } } % On construit un ensemble de dépendances
      sousProblemes ← sousProblemes ∪ { ensemble }
    FIN

    POUR prof ∈ profs FAIRE
      POUR préférence ∈ prof.préférences FAIRE
        SI préférence.groupeCours = groupeCours ALORS
          POUR pref ∈ prof.préférences FAIRE
            ens ← trouverEnsembleQuiContient(pref.groupeCours, sousProblemes)

            SI ens = {} ALORS
              ensemble ← ensemble ∪ { pref.groupeCours }
            SINON
              sousProblemes ← sousProblemes - { ens }
              ensemble ← ensemble ∪ { ens }
            FIN
          FIN
        FIN
      FIN
    FIN
  FIN

  RETOURNER sousProblemes
FIN
```

Pseudocode 3.3 Pseudocode de fragmentation du problème en plusieurs sous-problèmes sans inter-dépendance.

l'algorithme trouve une solution optimale dans un temps raisonnable. Heureusement, ces cas semblent être plutôt rares et il existe une solution permettant de contourner ces situations. Au lieu de se contraindre à traiter les deux groupes comme un seul, il est possible de les laisser séparés, mais en les évaluant deux fois : une fois où le professeur se voit attribuer le cours dont la préférence est critique, et l'autre fois, non. La figure 3.3 montre un exemple de ce genre de situation. Sur la gauche de la figure, le graphe de préférences de cours montre une préférence qui relie deux sous graphes (ou sous-problèmes) qui pourraient être indépendants. En évaluant deux fois le scénario — une fois en forçant l'assignation de la préférence critique, et l'autre fois sans son assignation —, il est possible de réduire considérablement le nombre de possibilités totales à évaluer. Lorsque l'évaluation des deux possibilités de scénario est terminée, il s'agit de conserver celle qui a permis la meilleure solution d'assignations.

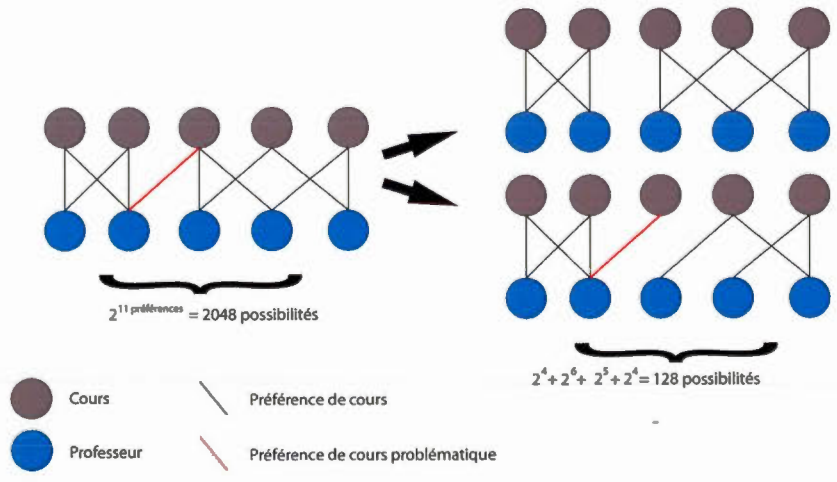


Figure 3.3 Fragmentation d'un problème en deux sous-problèmes, malgré une préférence critique.

Afin de trouver si une préférence entraîne un regroupement de deux sous-problèmes de taille importante, il s'agit simplement d'exécuter l'algorithme 3.3 plusieurs fois,

où à chaque fois une préférence différente est désactivée. Si, lors d'un retrait d'une préférence, la fonction retourne des ensembles plus petits, alors on est en présence d'une préférence critique et on peut agir selon la technique expliquée au paragraphe précédent.

Il est possible, quoique improbable, qu'un grand nombre de professeurs décident de fournir des préférences qui touchent à des catégories de cours différents (base de données, gestion de projets, électronique, ...) de telle sorte qu'il soit impossible de fragmenter le problème en plusieurs sous-problèmes rapides à traiter (voir section 4.5.3 pour détails). Dans ce cas, il n'y aura d'autre choix que d'utiliser un algorithme de recherche locale puisque l'espace de recherche restera trop vaste.

3.2.3 Le maintien de la propriété *arc-consistent*

Un raffinement qui permet d'accélérer le traitement consiste à conserver en tout temps le problème *arc-consistent*. Cette technique signifie que les domaines des variables devraient toujours contenir des valeurs encore assignables. Dans le contexte de l'assignation des tâches d'enseignement, il s'agit de vérifier, après chaque assignation, que les assignations suivantes sont encore possibles. Dans le cas où une préférence à venir est rendue impossible à assigner, il faut alors retirer de son domaine la valeur «assignée», ce qui la place automatiquement comme «non assignée». Concrètement, après chaque assignation et pour chaque future variable, on retire du domaine la possibilité «assignée» si le groupe-cours a déjà été assigné à quelqu'un d'autre, ou si le professeur ne désire pas davantage de tâche. Ce principe d'inférence se nomme *forward checking* [4][16].

Pour démontrer son fonctionnement, prenons l'exemple d'un professeur qui ne désire donner qu'une seule tâche et qui fournit 3 préférences de cours. Lorsqu'il se voit assigner sa première préférence de cours, son nombre de tâches souhaitées est alors mis à zéro et ses deux prochaines préférences sont automatiquement

placées comme «non assignée». Ceci découle du fait qu'il est impossible de lui assigner davantage de cours que le maximum demandé. Cette technique contribue à l'élimination de possibilités à explorer et permet de retourner un résultat plus rapidement.

3.2.4 L'utilisation du parallélisme

Puisque les ordinateurs possèdent de plus en plus de processeurs, une technique pour améliorer la performance de l'algorithme est de prendre avantage de leur puissance. La technique pour diviser un traitement à travers l'ensemble des processeurs disponibles se nomme le parallélisme. Il existe plusieurs types de parallélisme et, parmi ceux-ci, il existe plusieurs stratégies qui sont utilisées en fonction du traitement à faire.

Dans notre contexte, il est possible d'augmenter la performance du programme en répartissant les sous-problèmes à évaluer à travers les différents processeurs disponibles. Si, pour une session, il y a quatre sous-problèmes de même taille et que l'ordinateur utilisé possède quatre processeurs, alors chaque sous-problème peut être envoyé sur un processeur différent. Dans ce cas précis, l'algorithme serait quatre fois plus rapide, puisque tous les sous-problèmes seraient exécutés (traités) en même temps.

A priori, puisqu'il n'est ni possible de savoir quel ordinateur sera utilisé pour traiter l'algorithme (i.e. le nombre de processeurs disponibles), ni le nombre de sous-problèmes présents dans la session en cours, un fonctionnement par sac de tâches [20] est à préconiser. Cette technique, bien connue dans le monde du parallélisme, consiste à créer une collection de tâches à faire, où chacun des processeurs va piger des tâches à traiter tant qu'il y en a. Il est important de souligner qu'ici, le terme tâche signifie un traitement à faire, et non une tâche de cours pour un professeur. Par exemple, une tâche peut représenter le traitement d'un ensemble de groupes-cours à assigner.

3.2.5 Les cas de concurrences identiques

Il est possible, lors d'une session, que deux professeurs désirent donner les mêmes groupes-cours de telle sorte qu'il existe deux solutions optimales, c'est-à-dire ayant le même vecteur de points. Pour éviter un biais favorable à un professeur lors du traitement de l'algorithme, un ordonnancement aléatoire des variables pourrait être fait. Ceci empêcherait l'algorithme de toujours favoriser le même professeur lors de l'assignation des groupes-cours.

3.2.6 Le fonctionnement général

Prenant avantage des techniques expliquées dans les sections précédentes, l'algorithme que nous proposons a la forme décrite dans les étapes suivantes. En appliquant celles-ci, il est possible de simplifier suffisamment le problème pour permettre de trouver une solution optimale en quelques minutes maximum (voir chapitre 4 pour détails).

1. Assigner les assignations sans concurrence.
2. Fragmenter le problème en plusieurs sous-problèmes.
3. Trouver les préférences critiques, les retirer et refragmenter. Pour chaque préférence critique retirée, évaluer deux fois les scénarios : avec et sans leur assignation.
4. Paralléliser le traitement de chaque sous-problème et conserver ceux-ci *arc-consistents.*
5. Retourner la solution.

Dans l'optique de faciliter la compréhension du lecteur, le nom «RFPT» sera donné à l'algorithme, puisqu'il tente de **r**éduire la taille du problème, de le **f**ragmenter en plusieurs sous-problèmes et de **p**araller**l**iser son **t**raitement.

3.3 Le recuit simulé

Tel qu'expliqué précédemment, puisqu'il est possible que la fragmentation du problème en plusieurs sous-problèmes soit insuffisante pour permettre de trouver la solution optimale dans une période de temps raisonnable, le recuit simulé a été implémenté. Celui-ci représente donc une alternative disponible qui, étant donné un temps alloué (qui peut être plus ou moins court), permet de trouver une solution qui ne sera pas nécessairement la meilleure — au sens d'optimale — mais qui ne sera pas trop mauvaise.

Le recuit simulé est une métaheuristique qui permet de bien explorer un espace de recherche afin d'en ressortir une solution. Il fait donc partie des algorithmes approximatifs. Comparativement à certains algorithmes simples tels que *Hill Climbing* [19], le recuit simulé est capable de se déloger de maximum locaux (de bonnes solutions, mais pas nécessairement les meilleures), ce qui lui permet de mieux explorer l'espace de recherche et, idéalement, de trouver la solution optimale. Le pseudocode 3.4, tiré des travaux de Blum et Roli [3], décrit son fonctionnement.

Pour implémenter l'exploration de solutions via le recuit simulé, le pseudocode 3.5 a été utilisé. Celui-ci permet de trouver une solution voisine à celle actuellement conservée par cette métaheuristique. Le lecteur averti notera que la fonction prend avantage de l'algorithme *Hill Climbing*. Ceci permet, suite à la modification ou au retrait d'une préférence assignée, de maximiser les assignations des préférences de cours fournies par les professeurs. Cette étape est importante puisqu'il est possible que l'assignation d'un certain groupe-cours à un professeur donné réduise considérablement la qualité de la solution. Ceci est dû au fait que cette assignation engendrerait un effet domino sur les autres assignations de cours. En effet, elle pourrait empêcher un second professeur de donner ce groupe-cours alors que celui-ci est sa première préférence. Conséquemment, un autre groupe-cours d'une préférence moindre devra lui être assigné, ce qui rend ce dernier groupe-cours non disponible pour un troisième professeur qui le désire comme première préférence, et ainsi de suite. Il est donc nécessaire, lorsque le recuit simulé modifie l'état de la solution, de vérifier si d'autres assignations sont ainsi débloquées et attribuables.

```
% Note : La fonction f permet d'évaluer la qualité d'une solution.
s ← GénérerSolutionInitiale()
T ← T0              % Variable "température", réduite après chaque itération.
                    % Plus elle est réduite, moins l'algorithme accepte
                    % l'exploration d'un espace de recherche moins prometteur.
                    % La variable assure également de quitter la boucle,
                    % après un certain temps.

TANT QUE condition de fin n'est pas remplie FAIRE
  s' ← choisirAléatoirement(s)

  SI f(s') < f(s) ALORS          % La nouvelle solution est meilleure.
    s ← s'
  SINON
    s ← s' avec une probabilité de p(T, s', s)
  FIN

  MettreÀJour(T)
FIN
```

Pseudocode 3.4 Le recuit simulé, adapté de Blum et Roli [3].

```
FONCTION choisirAléatoirement( préférences: Set{Préférence},
                               solution: Set{Préférence} ): Set{Préférence}
  préférence ← choisirPreferenceAuHasard(préférences)

  SI préférence ∈ solution ALORS
    solution ← solution - { préférence }
  SINON
    pref ← trouverPréférence(préférence.groupeCours, solution)

    SI pref ≠ nil ALORS
      solution ← solution - { pref }
    FIN

    SI nbTaches(préférence) ≤ préférence.professeur.nombreTâchesVoulu ET
       sansConflitHoraire(solution, préférence) ALORS
      solution ← solution ∪ { préférence }
    FIN
  FIN

  appliquerHillClimbing(solution)
  RETOURNER solution
FIN
```

Pseudocode 3.5 Fonction permettant de trouver une solution voisine à celle actuellement trouvée.

3.4 Résumé

Dans ce chapitre, nous avons décrit l'algorithme RFPT. Il prend avantage de plusieurs techniques afin d'être en mesure de traiter le problème de l'assignation de tâches d'enseignement et de retourner une solution optimale. Dans le cas où, malgré les raffinements appliqués, l'espace de recherche reste trop volumineux pour être traité par l'algorithme RFPT, le recuit simulé représente une alternative intéressante pour trouver une solution de qualité. La prochaine section détaille certaines expérimentations que nous avons effectuées et fait une comparaison des résultats donnés par les deux algorithmes.

CHAPITRE IV

LES RÉSULTATS CONSTATÉS

Afin de tester l'algorithme RFPT introduit au chapitre 3 et d'être en mesure de constater sa capacité à résoudre le problème de l'assignation des tâches d'enseignement, des données de plusieurs sessions ont été nécessaires. Au départ, des données basées sur des sessions réelles ont été utilisées. Cependant, étant donnée la quantité limitée de données transmises par le département, il a été nécessaire d'en générer de nouvelles. C'est pourquoi un simulateur de sessions a été développé. Celui-ci permet de tester l'algorithme selon différents types de sessions, ce qui aide à comprendre dans quels contextes il est le plus performant et, inversement, les situations où il est inadapté. Ce chapitre détaille les simulations et fait le constat du rendement de l'algorithme.

4.1 Le traitement de sessions basées sur des données réelles

Les premiers tests qui ont servi à évaluer l'algorithme RFPT ont été faits à partir de données reçues du Département d'informatique de l'UQAM. Celles-ci représentent des sessions réelles et l'algorithme se devait de pouvoir les traiter. Le tableau 4.1 représente le sommaire des données recueillies ainsi que le temps qu'a pris l'algorithme à trouver le meilleur scénario d'assignations.

Comme le lecteur pourra le constater, ce faible nombre de sessions réelles démontre peu l'efficacité de l'algorithme. C'est pourquoi le simulateur de sessions a été créé.

Session	Nb. préf.	Nb. grp-cours	Nb. tâches	Nb. prof.	Temps exécution (sec.)
Hiver 2004	121	124	60	36	439
Aut. 2004	116	125	56.5	34	1366
Hiver 2005	114	117	52	35	8
Aut. 2015	96	90	53.5	30	7

Tableau 4.1 Sommaire des simulations basées sur des données réelles.

4.2 Le simulateur de sessions

Le simulateur de sessions a été développé comme une fonctionnalité à part entière du logiciel conçu dans le cadre de ce mémoire et prend la forme d'un créateur de sessions fictives. Voici les quatre paramètres qui doivent être spécifiés lors de la création d'une telle session :

- Le nombre de groupe-cours (par défaut = 110)
- Le nombre de tâches à assigner (par défaut = 55)
- Le nombre de professeurs (par défaut = 35)
- Le nombre de préférences de cours [1] (par défaut = 105)

Les valeurs par défaut des paramètres sont basées sur les données réelles (voir section 4.1). Pour chaque professeur, le nombre de tâches désirées et ses préférences de cours sont tous choisis aléatoirement.

4.3 L'environnement de simulation

La machine utilisée pour tester l'algorithme est un ordinateur doté d'un processeur Intel Core i7 4702MQ. Il possède 4 coeurs, considérés comme 8 par le système

1. Le nombre de choix faits par les professeurs.

d'exploitation étant donné la technologie *Hyper-Threading* d'Intel. L'ordinateur possède 8 Go de mémoire vive GDDR5. Puisque l'algorithme est mis en oeuvre en JavaScript, il faut ajouter que le navigateur choisi pour faire les tests est Google Chrome. Le choix du navigateur a été fait suite à plusieurs tests, où il a été possible de constater que Mozilla Firefox et Internet Explorer de Microsoft étaient plus lents à terminer l'exécution de l'algorithme. Aussi, Google Chrome offre une fonction native permettant de savoir le nombre de processeurs disponibles sur le poste de travail, ce qui est utile lorsque l'on veut paralléliser un traitement.

4.4 Les tests

Pour mieux constater le rendement de l'algorithme, des sessions fictives ont été générées. Celles-ci ont servi de test afin d'observer sa capacité à trouver des solutions (les attributions de tâches), et de vérifier la qualité de ses solutions par rapport aux solutions d'un autre algorithme. Puisque le recuit simulé est apte à se déprendre de maximum locaux et à bien explorer l'espace de recherche, il a été choisi comme algorithme de comparaison.

Chaque session fictive a été testée 5 fois par le recuit simulé, ce qui lui donne plus de chances de trouver une meilleure solution (sinon la meilleure). En ce qui concerne l'algorithme introduit au chapitre 3, il est déterministe et optimal. Ceci signifie que la solution qu'il retourne est nécessairement la meilleure et que son temps d'exécution ne varie pas pour une même session. C'est pourquoi il n'a été testé qu'une seule fois par session. Contrairement au recuit simulé, si le temps que prend l'algorithme pour trouver une solution devient trop long, il est possible de l'arrêter manuellement. Dans ce cas, aucun résultat n'est trouvé. Une limite de deux heures (7200 secondes) de traitement a été imposée pour l'algorithme RFTP.

En tout, 40 sessions fictives ont été créées. Chacune d'elles a une combinaison unique de paramètres introduits à la section 4.2. Le tableau 4.2 énumère tous les

Nb. préférences	55, 65, 75, 85, 95, 105, 115, 125, 135, 145
Nb. groupe-cours	60, 70, 80, 90, 100, 110, 120, 130, 140, 150
Nb. tâches	35, 40, 45, 50, 55, 60, 65, 70, 75, 80
Nb. professeurs	15, 20, 25, 30, 35, 40, 45, 50, 55, 60

Tableau 4.2 Les différents paramètres testés.

paramètres testés. Pour chaque session fictive, un seul paramètre n'est pas à sa valeur par défaut (voir section 4.2 pour les valeurs par défaut).

Il est évident que certaines sessions ne sont pas réalistes. Par exemple, il est peu probable, voire impossible, qu'une session n'offre que 60 groupe-cours disponibles pour l'ensemble des professeurs du Département d'informatique de l'UQAM. Ces sessions font néanmoins partie des tests afin de voir comment réagit l'algorithme dans ce genre de situation.

4.5 Les résultats

Les tableaux 4.3, 4.4, 4.5 et 4.6 présentent un sommaire des résultats obtenus lors des tests. Les cellules ayant la valeur «> 7200» signifient que l'algorithme RFPT n'a pu être complété en deux heures de traitement, ce qui rend la donnée non disponible. Cela dit, dans 75% des 40 sessions fictives testées, l'algorithme a été capable de retourner la solution optimale au problème d'assignation des tâches.

La variation du temps d'exécution de l'algorithme RFPT, qui peut être significative, s'explique par la taille de l'espace de recherche. Par exemple, il est possible de constater que plus il y a de groupe-cours disponibles, moins l'algorithme prend du temps à se compléter (voir tableau 4.3). Ceci est dû au fait que lorsqu'il y a une grande offre de groupe-cours, plusieurs professeurs sont en mesure d'avoir leurs premiers choix sans être en concurrence avec d'autres professeurs. Ceci permet de réduire le problème considérablement et réduit l'espace de recherche à

traiter par *branch and bound.* Aussi, le fait que certaines sessions prennent peu de temps à se compléter alors qu'une petite variation d'un paramètre l'augmente dramatiquement est simplement le fruit du hasard, puisque pour chaque session, les données sont générées aléatoirement. Certaines sessions sont donc plus faciles à traiter que d'autres. Par exemple, il a été constaté lors de certains tests que la fragmentation du problème en sous-problèmes n'est pas une étape suffisante pour réduire l'espace de recherche. Ce genre de situations survient lorsqu'une grande quantité de professeurs désirent donner les mêmes groupes-cours, ce qui a comme conséquence que l'algorithme de fragmentation (voir 3.3) est incapable de générer des graphes de groupes-cours suffisamment petits pour être traiter rapidement. Dans ce cas, les sous-problèmes générés contiennent encore une vaste partie de l'espace de recherche et la performance est affectée.

En excluant les sessions où l'algorithme n'est pas en mesure de trouver de solution à cause du temps d'exécution trop long, le temps médian nécessaire pour trouver la solution est de 11 secondes (voir figure 4.1 pour plus de détails). Ceci est acceptable et rejoint l'un des objectifs visés dans le cadre de ce mémoire.

Les sous-sections suivantes relèvent les points saillants observés lors de l'analyse des résultats des tests.

4.5.1 Les points forts

L'aspect le plus intéressant à souligner est le fait que les données des sessions antérieures obtenues du Département d'informatique de l'UQAM ont toutes pu être traitées par l'algorithme RFPT. Plus précisément, ce dernier a été en mesure de se compléter et de retourner une solution dans chacun des cas. Ceci semble donc suggérer qu'il est apte à traiter les données de sessions réalistes, mais il est difficile de s'en assurer, étant donné le faible nombre de scénarios testés. Il aurait été intéressant de faire de plus amples tests basés sur des données réelles, mais

	Algorithme RFPT		Recuit simulé	
Nb groupes-cours	Temps (sec)	Pourcentage de tests produisant un scénario optimal	Temps médian (sec)	Pourcentage de tests produisant un scénario optimal
60	> 7200	0%	4	n.d.
70	> 7200	0%	5	n.d.
80	73	100%	5	40%
90	55	100%	6	60%
100	55	100%	6	100%
110	24	100%	5	20%
120	16	100%	6	80%
130	4	100%	7	100%
140	7	100%	7	40%
150	9	100%	7	0%

Tableau 4.3 Sommaire des résultats obtenus en fonction du nombre de groupe-cours disponibles.

	Algorithme RFPT		Recuit simulé	
Nb tâches	Temps (sec)	Pourcentage de tests produisant un scénario optimal	Temps médian (sec)	Pourcentage de tests produisant un scénario optimal
35	968	100%	7	0%
40	22	100%	7	80%
45	2	100%	6	0%
50	697	100%	6	60%
55	7	100%	7	0%
60	245	100%	6	0%
65	40	100%	5	40%
70	9	100%	6	0%
75	449	100%	7	100%
80	11	100%	5	100%

Tableau 4.4 Sommaire des résultats obtenus en fonction du nombre de tâches à allouer.

	Algorithme RFPT		Recuit simulé	
Nb professeurs	Temps (sec)	Pourcentage de tests produisant un scénario optimal	Temps médian (sec)	Pourcentage de tests produisant un scénario optimal
15	> 7200	0%	6	n.d.%
20	> 7200	0%	6	n.d.%
25	1402	100%	7	40%
30	6878	100%	7	80%
35	136	100%	6	60%
40	58	100%	7	0%
45	3	100%	7	0%
50	11	100%	7	0%
55	3	100%	6	60%
60	7	100%	6	40%

Tableau 4.5 Sommaire des résultats obtenus en fonction du nombre de professeurs.

	Algorithme RFPT		Recuit simulé	
Nb préférences	Temps (sec)	Pourcentage de tests produisant un scénario optimal	Temps médian (sec)	Pourcentage de tests produisant un scénario optimal
55	2	100%	3	100%
65	2	100%	4	100%
75	2	100%	5	100%
85	5	100%	5	60%
95	3	100%	5	0%
105	1261	100%	5	0%
115	> 7200	0%	6	n.d.
125	> 7200	0%	8	n.d.
135	> 7200	0%	8	n.d.
145	> 7200	0%	9	n.d.

Tableau 4.6 Sommaire des résultats obtenus en fonction du nombre de préférences.

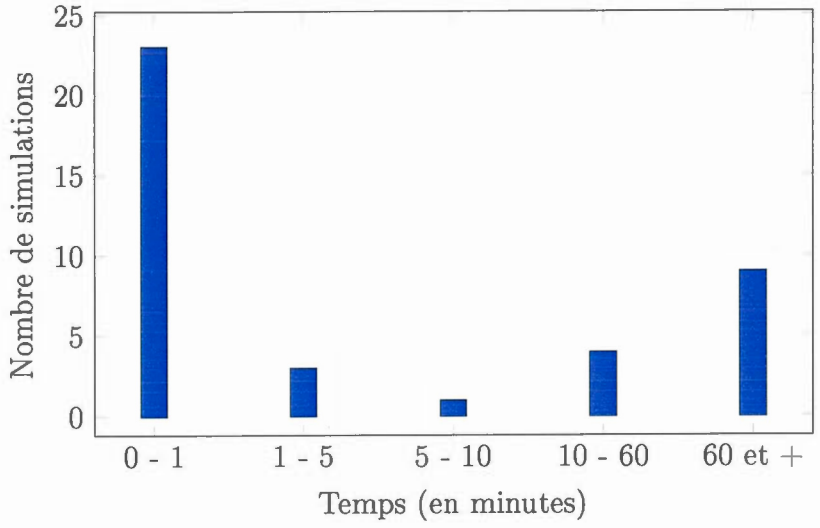


Figure 4.1 Temps d'exécution de l'algorithme RFPT, toutes simulations confondues.

puisque cela s'est avéré impossible d'en avoir davantage, des sessions fictives ont été utilisées.

Un autre point à soulever est le fait que la solution retournée par l'algorithme RFPT est toujours optimale, contrairement au recuit simulé. Il n'est nécessaire d'exécuter l'algorithme qu'une seule fois pour savoir quelle est la meilleure répartition des tâches d'enseignement à travers l'ensemble des professeurs. Ceci est donc un avantage par rapport au recuit simulé, qui n'est pas en mesure de garantir la qualité de sa solution.

4.5.2 Les points faibles

Comparativement au recuit simulé qui est en mesure de trouver une bonne solution en peu de temps, il est vrai que le traitement de la session automne 2004 a été assez long à se compléter par l'algorithme RFPT (23 minutes d'exécution). Deux raisons peuvent expliquer cette faible performance : le grand nombre de dépendances

entre les groupes-cours a nui à la fragmentation du problème et l'homogénéité des solutions a empêché l'algorithme de tronquer des branches de l'arbre de recherche.

Il est vrai que, dans 25% des cas, l'algorithme n'est pas en mesure de retourner de solution. Cela dit, il faut souligner le fait que les sessions problématiques sont, pour la plupart, peu réalistes. Le tableau 4.7 est une synthèse des situations où il est impossible de retourner une solution dans un temps raisonnable.

Situation	**Réalisme**	**Commentaire**
nb. professeurs ≤ 20	Peu réaliste	Il est difficile de concevoir une session avec plus de 100 préférences réparties sur si peu de professeurs.
nb. groupe-cours ≤ 70	Peu réaliste	Si l'offre de cours est trop petite, cela signifie que la plupart des professeurs veulent offrir les mêmes cours. À moins d'une diminution extrême du nombre d'étudiants inscrits en informatique à l'UQAM, il est peu probable qu'une session offre si peu de groupe-cours.
nb. préférences ≥ 115	Réaliste	L'espace de recherche devient trop grand pour l'algorithme, malgré les raffinements effectués.

Tableau 4.7 Session problématique pour l'algorithme.

L'analyse des résultats permet de constater que, lorsqu'il est difficile de tronquer l'espace de recherche pour une session donnée, le recuit simulé réussit mieux que l'algorithme RFPT. Par exemple, si 20 professeurs désirent offrir les mêmes groupe-cours et qu'ils ont tous la même priorité sur ces cours, l'algorithme ne sera alors pas en mesure de couper l'espace de recherche et devra évaluer toutes les

possibilités d'assignation, ce qui demande du temps. Par contre, dans ces cas, le recuit simulé trouvera probablement une solution optimale.

L'algorithme est apte à traiter un espace de recherche assez vaste (un grand nombre de préférences), surtout lorsqu'il est en mesure de tronquer une partie de l'espace de recherche jugé non optimal. Une des façons d'y parvenir est en identifiant les coordonnateurs de cours, les nouveaux professeurs, le directeur et les mauvaises évaluations. En ajoutant ces informations, l'algorithme est plus apte à réduire l'espace de recherche, étant donné la plus grande variabilité de la qualité des scénarios trouvés.

4.5.3 Remarques sur la fragmentation

La fragmentation augmente incontestablement la performance de l'algorithme en terme de temps. Cependant, dans le cas où la fragmentation est possible par le retrait d'une préférence, elle a un coût. Au départ, l'algorithme a demandé du réglage entre la nécessité de trouver les préférences critiques servant à la fragmentation et le besoin de trouver la solution rapidement. En effet, après plusieurs tests effectués, l'algorithme est en mesure de trouver jusqu'à trois préférences critiques dans un temps raisonnable. Autrement, la durée que prend cette recherche explose puisque le temps nécessaire pour trouver les bonnes préférences à retirer est exponentiel. L'algorithme commence par tenter le retrait d'une seule préférence et constate si l'espace peut être fragmenté en deux sous-problèmes suffisamment petits. Dans le cas où l'espace reste trop vaste, l'algorithme tente alors le retrait d'une deuxième préférence, et ainsi de suite. Précisément, ceci correspond à p^n évaluations, où p est le nombre de préférences et n le nombre de préférences qui peuvent être retirées. Actuellement, et pour une session contenant une centaine de préférences, l'algorithme peut prendre plus d'une heure à évaluer toutes les fragmentations possibles par le retrait de quatre préférences. C'est pourquoi le nombre maximal de préférences pouvant être retirées a été réglé à trois.

Le retrait d'une préférence critique rend le problème plus simple, mais double le nombre de scénarios à évaluer. En effet, lors du retrait d'une préférence, il faut nécessairement traiter deux scénarios : avec l'assignation forcée de la préférence, et sans son assignation. Conséquemment, si deux préférences sont retirées, il y a 2×2 scénarios à évaluer. À trois préférences, l'algorithme doit évaluer $2 \times 2 \times 2$ scénarios, et ainsi de suite. Ceci peut être résumé à 2^p scénarios, où p est le nombre de préférences retirées. Il y a donc un équilibre entre le nombre de préférences que l'on tente de retirer et la quantité de scénarios à évaluer.

CHAPITRE V

LE LOGICIEL DÉVELOPPÉ POUR AIDER À L'ASSIGNATION DES TÂCHES D'ENSEIGNEMENT

Une partie de ce mémoire a été consacrée à la conception d'un logiciel d'attribution des tâches d'enseignement, qui est disponible sur demande. Celui-ci a été développé dans le but de faciliter le travail de la personne responsable de l'assignation des tâches. Afin de bien modéliser et implémenter cet outil, il a été élaboré en se basant sur un document créé par une personne — notre directeur de recherche — qui a participé à l'attribution des tâches pendant plusieurs années, puisqu'il était directeur du Département d'informatique. Ce document (voir appendice B) détaille les cas d'utilisation que devait contenir le logiciel, décrit les différents acteurs impliqués et explique, de façon générale, le fonctionnement souhaité du logiciel. Il a donc servi de document d'exigences et a permis de cerner les fonctionnalités à implémenter afin de bien répondre aux besoins du département. Ce chapitre fait état de l'architecture et des fonctionnalités du système développé dans le cadre de ce mémoire.

5.1 L'architecture

Le logiciel a été implémenté avec les technologies du Web. Plus précisément, la structure interne est basée sur Ruby on Rails [1], la persistance des données est

1. http://rubyonrails.org

assurée par une base de données SQLite [2] et l'aspect présentation est conçu à l'aide de HTML, JavaScript et de feuilles de styles CSS. Le choix d'utiliser Rails est dû à sa philosophie, qui dicte l'utilisation d'une convention de programmation stricte (en anglais : *convention over configuration*). Ceci signifie que la programmation doit être faite selon la convention de Rails, et qu'il n'est donc pas possible de dévier de leurs standards. En acceptant la convention de Rails, le code est normalisé, il est plus compréhensible par les pairs et la configuration requise au fonctionnement du logiciel est simplifiée. Puisque ce dernier pourrait être repris par un prochain étudiant dans une étude future, il a été naturel de se diriger vers un environnement de développement comme celui-ci. Concernant le système de gestion de versions, git [3] a été utilisé. La raison du choix de cet utilitaire plutôt qu'un autre (Perforce, SVN, etc.) est basé sur sa popularité, sa gratuité, son aspect distribué et la facilité avec laquelle il est possible de collaborer entre développeurs. Finalement, une série de tests unitaires et d'intégrations ont été effectués pour vérifier que le système fonctionne correctement.

En ce qui concerne les interfaces graphiques, plusieurs librairies ont été utilisées. La plus importante est Bootstrap [4], car elle aide à la disposition de plusieurs composantes visuelles sans avoir à y mettre beaucoup d'effort. jQuery [5] est également une librairie qui a été utilisée, puisqu'elle permet de modifier dynamiquement les pages Web. Cette dernière possède également plusieurs fonctions d'animation, ce qui améliore l'esthétique de l'application et l'expérience utilisateur.

Finalement, l'implémentation des algorithmes a été faite en JavaScript. Ceci permet de diminuer la charge du serveur et de rediriger le traitement des tâches

---

2. https://www.sqlite.org

3. https://en.wikipedia.org/wiki/Git_(software)

4. http://getbootstrap.com

5. https://jquery.com

lourdes vers les machines clientes. Dans ce cas-ci, le navigateur est donc l'endroit où sont exécutés les algorithmes. Cette décision a été prise afin de ne pas surcharger le serveur lorsque l'algorithme est en cours d'exécution. Autrement, cette charge ralentirait le temps de réponse du serveur, ce qui aurait des répercussions sur les autres usagers essayant d'avoir accès au logiciel ou à tout autre application Web hébergée sur ce même serveur. Les figures 5.1, 5.2 et 5.3 sont trois impressions écran tirées du logiciel.

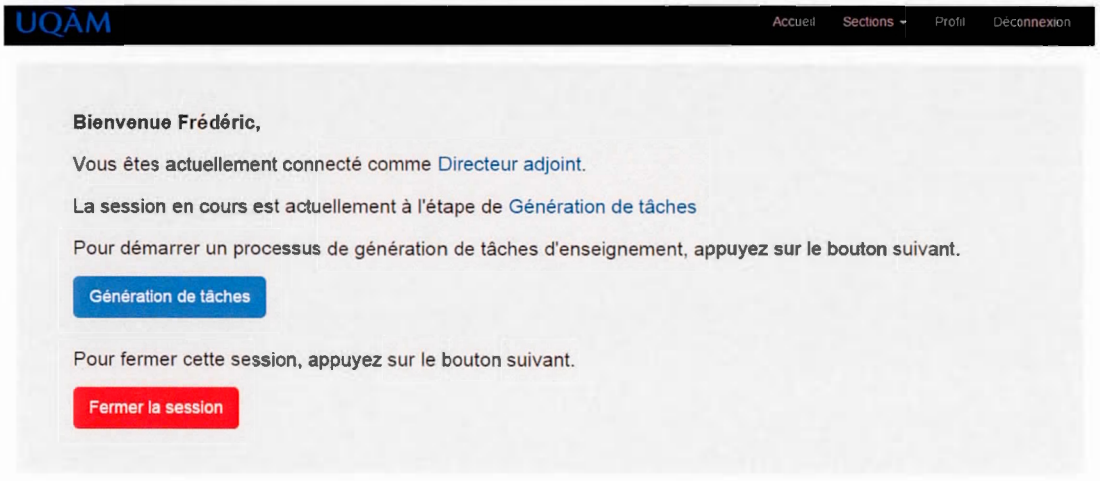


Figure 5.1 Page d'accueil du directeur adjoint.

5.2 Les fonctionnalités

Actuellement, le système possède toutes les fonctionnalités nécessaires à la génération de scénarios de répartion des tâches. Plus précisément, il est en mesure de faire les tâches suivantes :

1. La gestion des usagers. [6]

6. Par «gestion», on entend : créer, modifier, supprimer.

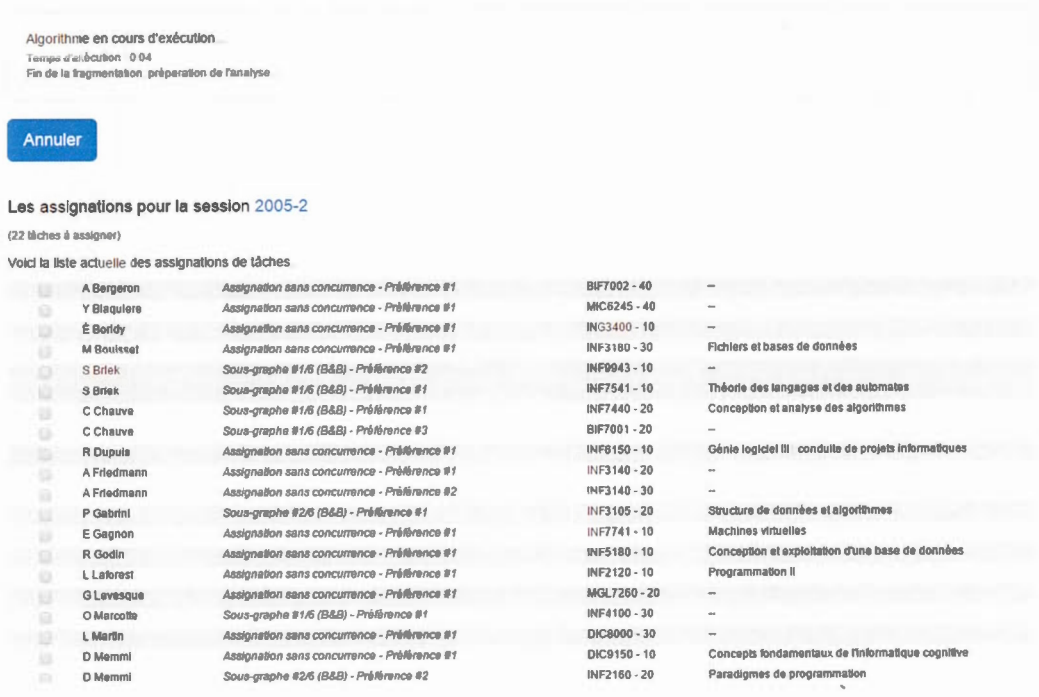


Figure 5.2 Page de traitement (exécution de l'algorithme en cours).

2. Le contrôle des accès aux fonctionnalités, via un module de connexion et de déconnexion, et utilisant une notion de «rôle utilisateur».
3. La gestion des sessions de cours.
4. La gestion des cours, des groupes-cours et de leur horaire.
5. La gestion des programmes gradués, des coordonnateurs de cours, et des mauvaises évaluations.
6. La gestion des choix de cours des professeurs.
7. Le déclenchement des algorithmes, puis la gestion des assignations de cours des professeurs.
8. La visualisation et l'exportation des assignations sauvegardées.

Ces fonctionnalités ont été implémentées avec l'objectif d'avoir un système suffisamment développé pour être utilisé par l'ensemble des professeurs du Département d'informatique de l'UQAM. Il a été conçu pour faciliter la cueillette des

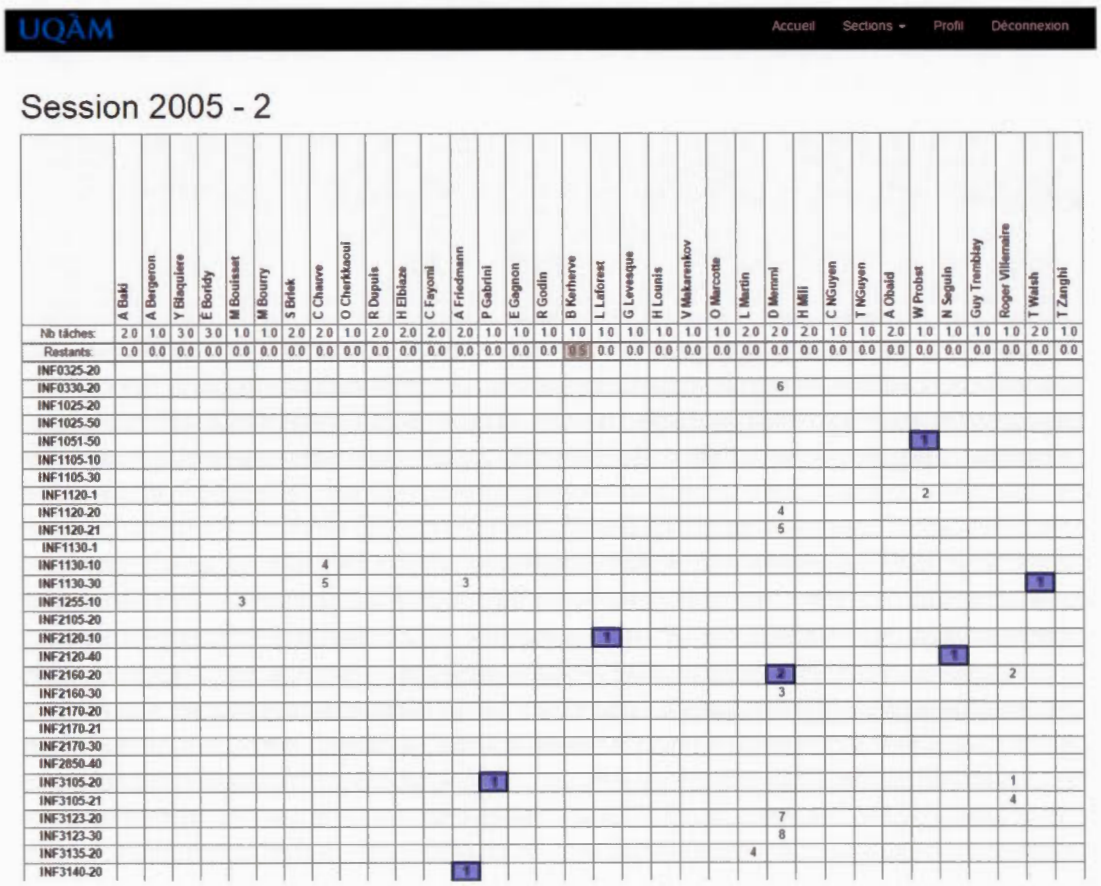


Figure 5.3 Exemple de tableau d'assignations.

préférences de choix de cours des professeurs et pour aider à trouver le meilleur scénario d'assignation de tâches en fonction des préférences spécifiées.

5.3 La gestion des accès par l'assignation de rôles aux usagers

Afin de contrôler l'accès aux fonctionnalités et aux données du système, celui-ci utilise une stratégie par «rôle», où chacun de ceux-ci contient un ensemble de droits sur le logiciel. Chaque usager possède donc un rôle, ce qui lui donne une série d'accès en fonction de ce qu'il doit accomplir dans le système. Voici les rôles disponibles et leurs droits associés :

1. *Directeur adjoint* : Est le gestionnaire des sessions et de leur déroulement (voir section 5.4 pour détails). Il peut également modifier les préférences de cours des autres professeurs. Finalement, il est responsable du déclenchement de l'algorithme et de l'assignation des tâches d'enseignement.
2. *Assistant administratif* : Est en mesure de gérer les professeurs, les cours, les groupes-cours (et leur horaire), les coordonnateurs et les programmes gradués
3. *Professeur* : Est en mesure de définir le nombre de tâches souhaitées pour la prochaine session, de spécifier ses préférences de choix de cours et de modifier son profil (mot de passe, nom, courriel)
4. *Inactif* : Ne possède aucun droit sur le système (rôle attribué, par exemple, à un ancien professeur)

Un rôle spécial peut également être attribué à tout usager. Ce rôle, nommé «administrateur système», permet d'avoir accès à toutes les fonctionnalités du système. Il peut être donné à un responsable chargé du développement et de la maintenance du logiciel ou à tout autre individu dont la tâche est de superviser l'état du logiciel et le déroulement des sessions.

5.4 Le déroulement d'une session

Lors d'une visite sur l'application Web, les fonctionnalités disponibles du logiciel varient non seulement en fonction du rôle de l'usager, mais en fonction de l'état d'une session également. Par exemple, il est impossible de choisir ses préférences de cours si aucune session n'est actuellement active.

Lorsqu'une nouvelle session est créée, elle passe par six étapes distinctes (voir figure 5.4). Les sous-sections suivantes expliquent ces étapes et ce qui doit être fait pour chacune d'elles.

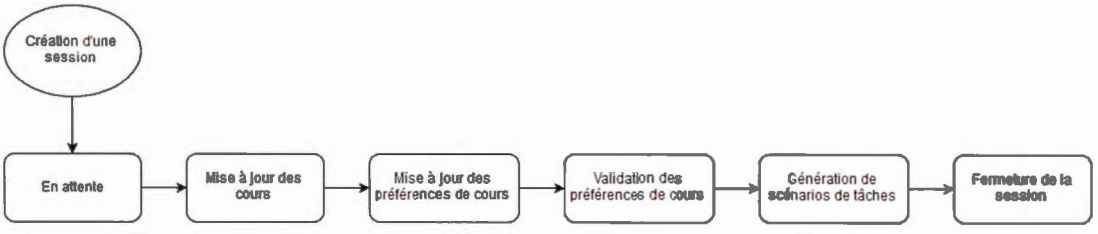


Figure 5.4 Les six étapes d'une session.

5.4.1 En attente

Cette étape initiale est utilisée pour signifier que rien n'est à faire, et qu'aucune intervention des usagers n'est requise. C'est une étape transitoire entre la création d'une session et le début du processus actif. L'impression écran 5.5 montre l'écran d'accueil du professeur lorsqu'aucune session n'est active. Lorsque le temps est venu, le directeur adjoint peut passer à la prochaine étape et déclencher le processus actif de la session.

5.4.2 La mise à jour des cours

Cette première étape active consiste à mettre les cours, les groupes-cours (et les horaires), les professeurs, les coordonnateurs et les programmes gradués à jour. Ceci peut être fait par l'assistant administratif ou encore par le directeur adjoint (voir impression écran 5.6).

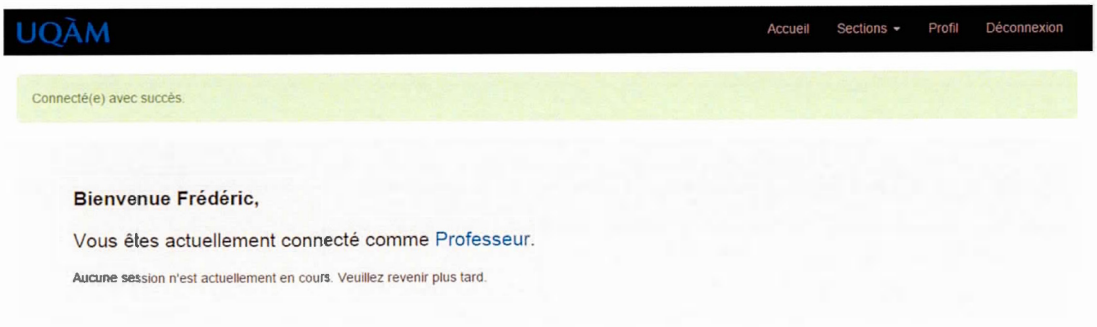


Figure 5.5 Aucune session active.

5.4.3 La mise à jour des préférences de cours des professeurs

Cette étape s'adresse aux professeurs qui doivent spécifier le nombre de tâches souhaitées ainsi que leurs préférences de cours (choix de groupes-cours) pour la session. Pour ce faire, les professeurs peuvent utiliser une page Web prévue à cet effet (voir impression écran 5.7).

5.4.4 La validation des préférences

Cette étape permet au directeur adjoint de vérifier les préférences entrées par l'ensemble des professeurs, et de les modifier, au besoin (voir impression écran 5.8). Ceci peut être pratique dans le cas où, par exemple, un professeur a fourni oralement au directeur ses préférences de cours.

5.4.5 La génération de scénarios d'assignations de tâches

Cette étape permet au directeur adjoint de démarrer l'algorithme RFPT ou encore le recuit simulé afin de générer un scénario d'assignation de tâches. Lors de cette étape, il est possible de faire des assignations manuelles et de recommencer les

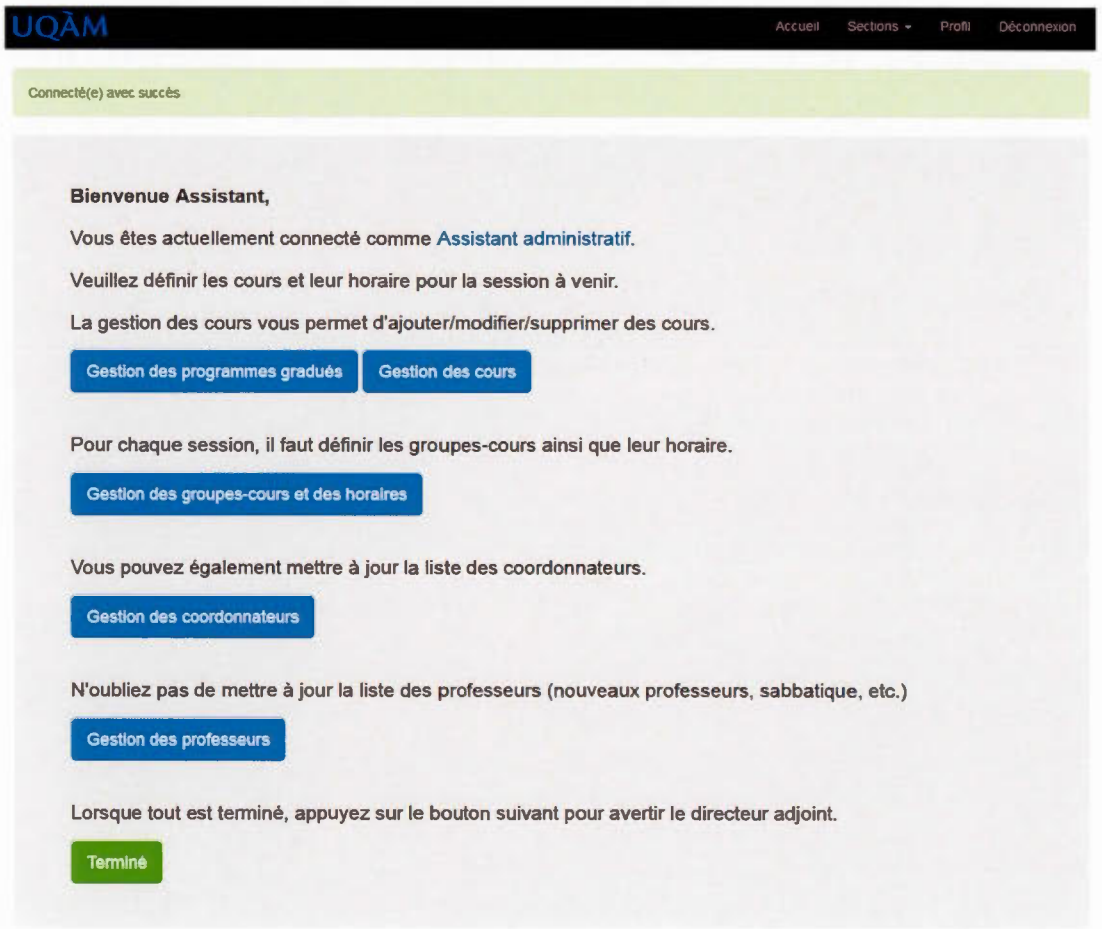


Figure 5.6 Session à l'étape de la mise à jours des cours.

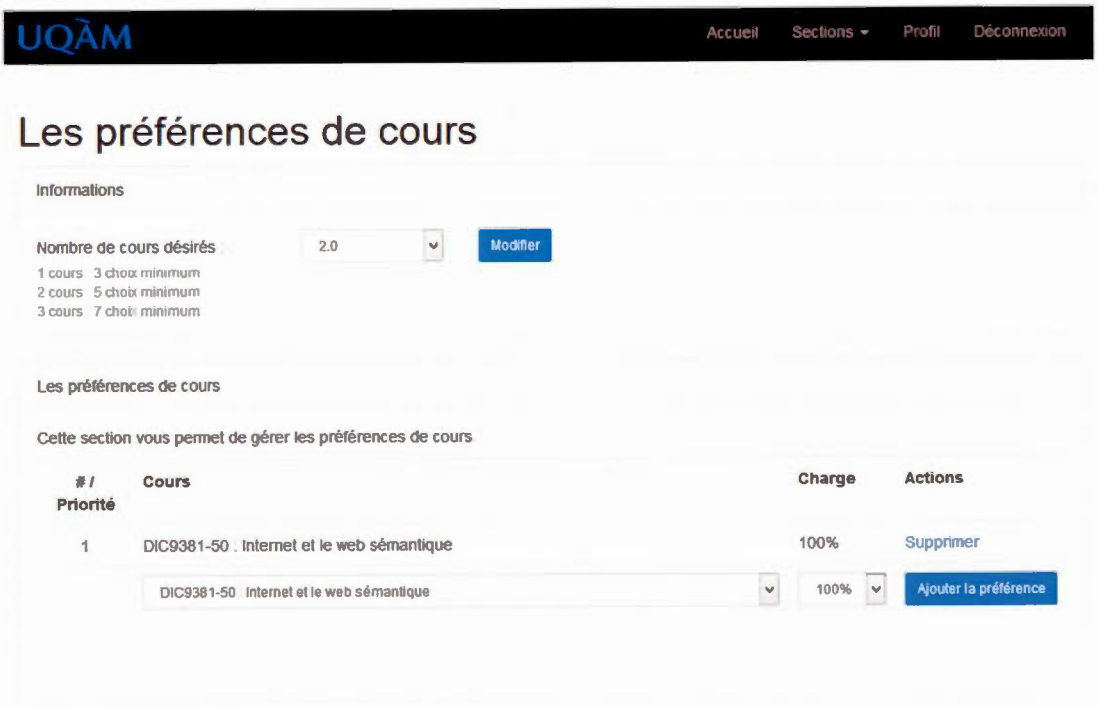


Figure 5.7 Session à l'étape de la mise à jour des choix de cours des professeurs.

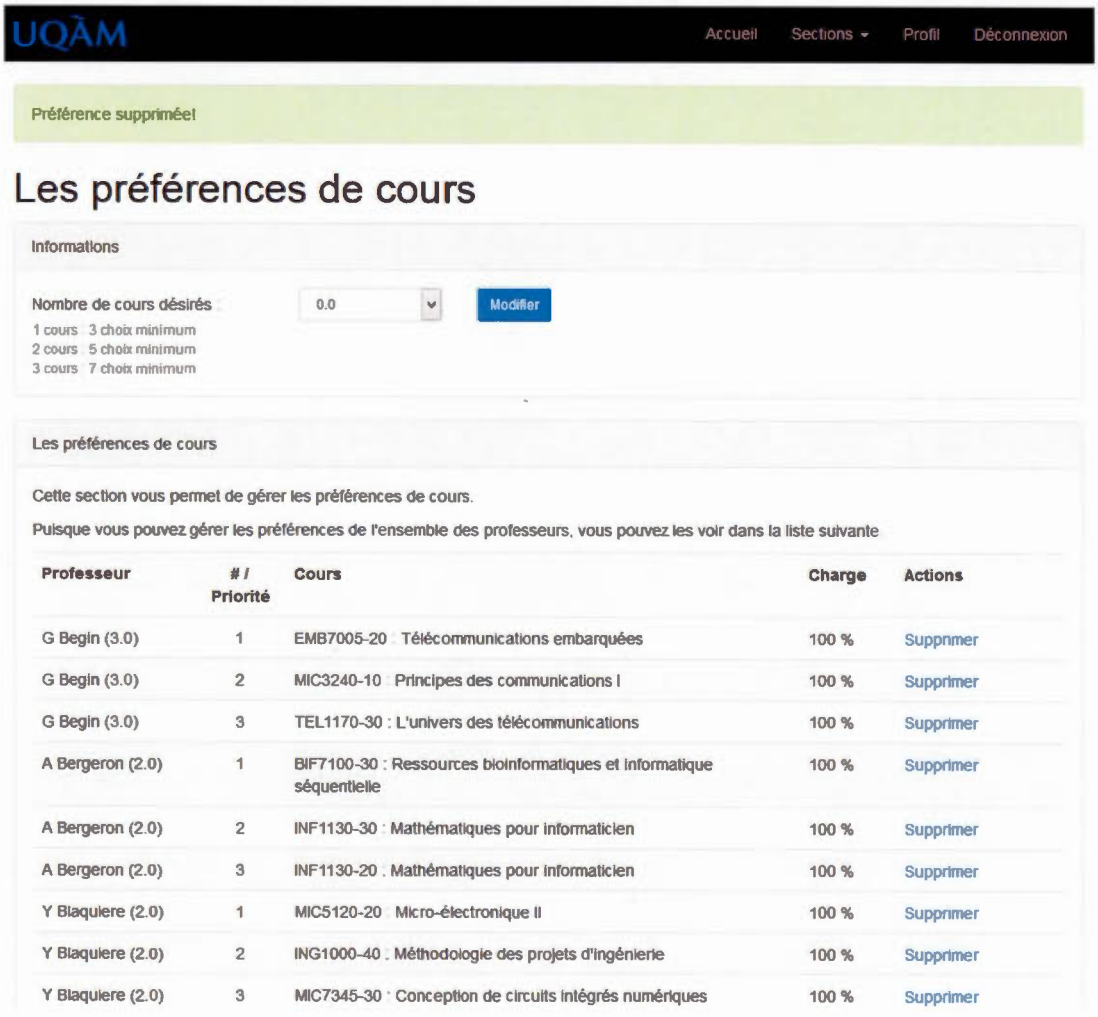


Figure 5.8 Session à l'étape de la validation des préférences.

algorithmes à plusieurs reprises (voir impression écran 5.9). Cette approche itérative permet de raffiner la recherche. Par exemple, suite à l'assignation forcée (i.e. manuelle) d'une tâche à un professeur, le scénario retourné par RFPT pourrait être significativement différent, puisqu'il n'aurait plus à traiter ni le groupe-cours, ni la tâche de ce professeur. Lorsque le scénario retourné est jugé satisfaisant, le directeur adjoint est en mesure de le sauvegarder via un bouton prévu à cet effet.

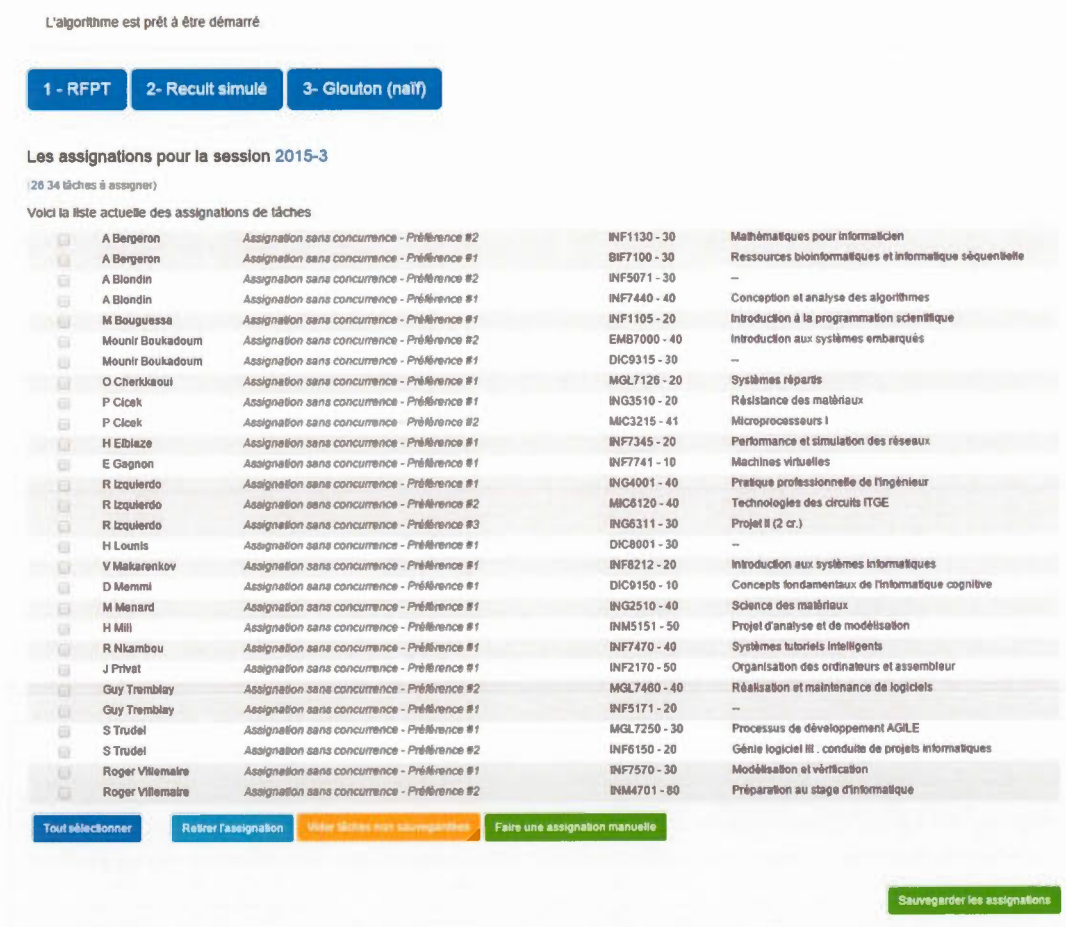


Figure 5.9 Session à l'étape de la génération de scénarios de tâches.

5.4.6 La fermeture d'une session

Lorsque le scénario de répartition des tâches est sauvegardé, il est possible de mettre la session en statut «complétée». Cette dernière étape ferme la session et les données sont ainsi conservées comme références futures. Aucune activité particulière n'est à faire lors de cette étape.

CONCLUSION

Ce mémoire avait comme objectif de maximiser l'assignation des tâches d'enseignement des professeurs du Département d'informatique de l'UQAM en respectant les règles départementales (cf. Appendice A). Comparativement à la recherche antérieure réalisée par Xia en 2006, qui utilise une approche basée sur des algorithmes approximatifs, le présent mémoire a tenté d'élaborer une résolution du problème via l'utilisation d'un algorithme complet.

En considérant le problème comme étant de type CSP, il a été possible de le simplifier via l'utilisation de plusieurs techniques, ce qui le rend moins complexe à résoudre. En effet, l'allocation des assignations sans concurrence, suivi d'une fragmentation du problème en plusieurs sous-problèmes traités parallèlement par l'algorithme *branch and bound* a permis de résoudre une grande partie des sessions testées. Plus précisément, malgré le nombre limité de données obtenues du département, les sessions basées sur des données réelles ont toutes pu être solutionnées par cette approche, ainsi qu'un bon nombre des sessions fictives.

Comparativement à l'utilisation d'un algorithme approximatif, l'approche préconisée pour ce travail n'a pas été en mesure de résoudre les sessions fictives ayant des données extrêmes. Par exemple, lorsque le nombre de préférences explose, la compétition pour chaque groupe-cours devient si énorme que l'algorithme n'est pas en mesure de se compléter dans un temps raisonnable (moins de deux heures). Pour y remédier, plusieurs pistes peuvent être explorées. Par exemple, il serait possible de se rabattre sur un algorithme approximatif dans de tels cas, ou encore de tester l'ordonnancement dynamique des variables en privilégiant certaines assignations problématiques afin de déterminer si cela contribue à diminuer le temps

d'exécution. Une option intéressante serait de lancer, après un certain temps, le recuit simulé à partir du meilleur résultat actuellement trouvé par RFPT.

Afin d'évaluer davantage les performances de l'algorithme RFPT, certaines avenues restent à être explorées. Par exemple, il serait intéressant de faire l'évaluation de sessions fictives dont les préférences de groupes-cours ne sont pas complètement aléatoires, mais regroupées selon les domaines d'expertise des professeurs (la gestion de projets, les bases de données, l'électronique, etc).

Finalement, le logiciel réalisé dans le cadre de ce mémoire est un outil qui reste à être testé et évalué par le département. Le logiciel a été implémenté afin de simplifier le travail lié au problème de l'assignation des tâches d'enseignement, en partant de l'entrée des préférences de groupes-cours par chaque professeur jusqu'au calcul du scénario optimal des répartitions à travers le département.

APPENDICE A

RÈGLES DÉPARTEMENTALES

AVIS DE PROPOSITION
Département d'informatique
Université du Québec à Montréal

POLITIQUE RELATIVE À L'ATTRIBUTION DES CHARGES D'ENSEIGNEMENT

PROPOSITION Nº R-11-740
Sur proposition du Comité exécutif ;

ATTENDU la lourde tâche de travail associée au poste de Directeur de département;

ATTENDU que le directeur de département doit malgré tout enseigner deux (2) cours par année ;

ATTENDU qu'un directeur de département pourrait difficilement assumer toutes les tâches qui lui incombent et devoir en plus développer un nouveau cours ou enseigner un cours qu'il n'aurait jamais donné ;

Il est proposé de modifier comme suit la politique départementale d'attribution des charges d'enseignement, les règles 2-8 étant appliquées dans l'ordre indiqué :

Choix des professeurs :

1. Chaque professeur doit normalement soumettre un nombre minimum de choix de groupes-cours spécifié comme suit:
- Pour enseigner 1 cours : 3 choix
- Pour enseigner 2 cours : 5 choix
- Pour enseigner 3 cours : 7 choix

Règle relative au directeur de département :

2. Nonobstant la règle 1, le Directeur de département peut ne soumettre qu'un seul choix, lequel lui sera accordé. Cette règle ne s'applique toutefois que pour deux trimestres par année académique.

Règles relatives à la qualité de l'enseignement :

3. Sur recommandation du comité de soutien pédagogique, un professeur qui aurait reçu à deux reprises pour deux ou trois prestations successives d'un cours donné une lettre indiquant une évaluation insatisfaisante pour ce cours pourra se voir refuser l'attribution de ce cours pour une période de six sessions suivant la dernière attribution.
4. Dans le cas de cours d'études avancées, une priorité sera accordée aux nouveaux professeurs et aux professeurs activement engagés dans l'encadrement d'étudiants du programme commanditaire principal du cours et l'attribution sera faite en consultation avec le directeur du programme commanditaire (article 10.24 de la convention collective).

Règles relatives à l'attribution selon le choix des professeurs :

5. En conformité avec la politique départementale qui veut qu'un coordonnateur donne le cours coordonné au moins une fois par an, le professeur coordonnateur d'un cours à groupes multiples a priorité pour le choix de son groupe ou pour le choix de son premier groupe s'il demande plus d'un groupe du même cours, toutefois ce groupe devra faire partie de ses deux premiers choix.
6. Tout professeur devrait se voir attribuer au moins un de ses deux premiers choix.
7. Un professeur qui donne un cours pour une première fois a priorité pour redonner ce cours deux autres fois subséquemment, et perd ensuite sa priorité. Un professeur qui donne de nouveau un cours qu'il a déjà donné dans le passé a priorité pour le redonner une autre fois subséquemment, mais perd ensuite sa priorité.
8. Un professeur qui se voit attribuer un cours aux études avancées n'a pas de priorité pour d'autres cours aux études avancées.

ADOPTÉE À L'UNANIMITÉ

Guy Tremblay
Directeur
Département d'informatique
G.T./sd

APPENDICE B

DOCUMENT D'EXIGENCES DU LOGICIEL

Logiciel d'attribution des tâches d'enseignement
Description des exigences

Guy Tremblay

Juin 2014

Table des matières

1 Vision

L'attribution des tâches d'enseignement aux professeurs du département est un processus qui doit se faire à chaque trimestre (été, automne et hiver), et ce plusieurs semaines/mois avant le début du trimestre. Ce processus d'attribution peut être relativement complexe à cause des diverses règles qui doivent être respectées. Ces règles sont décrites dans la résolution départementale R-11-740 intitulée "Politique relative à l'attribution des charges d'enseignement".

Actuellement, tout est fait "à la main" : les professeurs indiquent leurs choix sur des formulaires papier, ces choix sont entrés manuellement dans un chiffrier par l'assistante administrative, puis le directeur adjoint à l'enseignement utilise le chiffrier pour, toujours "à la main", procéder à l'attribution des tâches en respectant les diverses règles départementales.

L'objectif de ce projet est de développer un logiciel pour supporter ce processus et aider l'exécutif du département d'informatique — plus particulièrement le directeur adjoint à l'enseignement — à effectuer l'attribution des tâches d'enseignement aux professeurs du département.

Une particularité des règles d'attribution est qu'étant donné les choix indiqués par les professeurs, il est possible *qu'il n'existe aucune solution satisfaisant l'ensemble des règles.* Dans certains cas, on doit donc plutôt chercher à obtenir une attribution des tâches *qui contrevienne au moins de règles possibles.*

De plus, de nombreux changements surviennent durant le procesus — discussions avec les collègues ou les directeurs de programme, modifications d'horaire pour permettre certaines attributions de tâches, etc. Il est donc crucial que le logiciel puisse fonctionner *de façon itérative* : une première version de l'attribution est produite, des modifications y sont apportées — qui peuvent contrevenir à certaines règles de la politique départementale ou, même, qui vont à l'encontre des conflits d'horaires (dans certains cas, on modifie l'horaire du cours *après* l'attribution, etc. — dans le but de produire une attribution révisée, etc.

2 Acteurs

Acteur	**Objectif**
Administrateur	Configurer le logiciel
Assistante administrative	Indiquer les cours disponibles et leurs horaires Indiquer l'attribution finale une fois la session effectivement amorcée
Directeur adjoint	Gérer le statut des professeurs Vérifier les choix des professeurs Produire une première attribution des tâches d'enseignement Modifier une attribution pour l'améliorer
Professeur	Indiquer ou modifier ses choix de cours

3 Description générale du "processus d'affaires"

Le processus d'attribution des tâches d'enseignement démarre plusieurs semaines avant le début d'une session. Ces processus comporte plusieurs étapes. Dans ce qui suit, celles indiquées par (*) ne sont pas liées au présent logiciel mais sont simplement indiquées pour bien comprendre l'ensemble du processus.

1. Le directeur adjoint à l'enseignement indique le début d'une nouvelle session d'attribution et il met à jour le statut des professeurs pour cette session (congés sabbatique, coordination de cours, direction du département, etc.).
2. L'assistante administrative indique les divers cours qui sont offerts avec leurs horaires.
3. Chaque professeur-e indique les cours qu'il–elle désire enseigner.

   ...
4. Le directeur adjoint vérifie les choix des professeurs.
5. Le directeur adjoint fait produire une première ébauche de l'attribution des tâches.
6. Le directeur adjoint raffine l'attribution des tâches.

   ...
7. L'exécutif produit une résolution pour l'attribution des tâches. (*)
8. L'assemblée départementale approuve l'attribution des tâches. (*)
9. L'assistante administrative entre dans ACCENT les tâches des professeurs. (*)
10. Après le début de la session (pour être certain qu'il n'y aura plus de modification), l'assistante administrative indique l'attribution finale et effective des cours pour fins de l'historique.

4 Cas d'utilisation

Remarques :

- Dans tous les cas qui suivent, on suppose que l'utilisateur s'est correctement identifié (code MS standard UQAM).

4.1 Le directeur adjoint lance l'attribution pour une nouvelle session

Précondition : Aucune.

	Directeur adjoint		**Système**
1.	Le directeur adjoint demande d'amorcer le processus d'attribution pour une nouvelle session.	2.	Retourne l'identification de la session (modifiable par l'administrateur) ainsi que la liste des professeurs et leur statut.
3.	Le directeur adjoint modifie le statut de certains professeurs.	4.	Retourne la liste modifiée des professeurs et leur statut.
5.	Le directeur adjoint confirme que tout est ok.	6.	Sauvegarde les informations.

4.2 L'assistante administrative indique (ou modifie) les cours qui sont à l'horaire

Précondition : Le directeur adjoint a lancé le processus d'attribution.

	Assistante administrative		**Système**
1.	L'assistante administrative indique la session qu'elle veut traiter.	2.	Retourne la liste des cours pour cette session, qui peut être vide.
3.	L'assistante administative modifie la liste des cours ainsi que leurs horaires.	4.	Retourne la liste modifiée.
5.	L'assistante administrative confirme que la liste est ok.	6.	Sauvegarde les informations.

4.3 Un professeur indique (ou modifie) ses choix de cours

Précondition : L'assistante administrative a indiqué les horaires de cours.

	Professeur		Système
1.	Le professeur modifie ou indique ses choix.	2.	Signale différents "cas d'erreur" — par exemple, cours en conflit d'horaire, nombre insuffisant de choix pour le nombre de cours désirés — mais sans nécessairement bloquer le processus.
3.	Le professeur confirme ses choix.	4.	Sauvegarde les choix, en retournant à nouveau un message si certaines règles ne sont pas respectées.

4.4 Le directeur adjoint vérifie les choix des professeurs

Précondition : L'assistante administrative a indiqué les horaires de cours, les professeurs en ont été informés et du temps a été laissé aux professeurs pour faire leurs choix.

	Directeur adjoint		Système
1.	Le directeur demande la liste des professeur et leurs choix.	2.	Retourne la liste des professeurs. Ceux qui n'ont pas fait de choix ou qui ont fait des choix avec des "erreurs" sont pré-sélectionnés et un message spécifique et prédéfini est associé à chacun.
3.	Le directeur confirme les cas à corriger et/ou en ajoute d'autres, en modifiant si nécessaire certains messages. Si aucun cas erroné ou incomplet, alors on le confirme.	4.	Un courriel est transmis aux professeurs concernés leur demandant d'apporter des corrections.

4.5 Le directeur adjoint produit une première attribution départementale des cours

Précondition : Le directeur adjoint a validé les choix des professeurs.

	Directeur adjoint		Système
1.	Le directeur adjoint demande la production d'une première attribution.	2.	Retourne l'attribution qui satisfait le plus de règles — ou qui contrevient au moins de règles possibles.

4.6 Le directeur adjoint révise une attribution départementale des cours

Précondition : Le directeur adjoint a fait produire une première attribution des tâches.

	Directeur adjoint		Système
1.	Le directeur adjoint demande la liste des attributions départementales.	2.	Retourne la liste des attributions.
3.	Le directeur adjoint sélectionne une attribution départementale.	4.	Retourne les détails de l'attribution (contenu, niveaux de qualité, etc.)
5.	Le directeur modifie une attribution particulière pour un professeur.	6.	Retourne l'attribution départementale modifiée.
7.	...	8.	...
9.	Le directeur demande la sauvegarde de l'attribution départementale.	10.	L'attribution départementale est sauvegardée.

Remarques :

- L'idée sous-jacente est d'avoir accès à différents scénarios d'attribution, qui pourraient alors être comparés quant à leur effets et niveaux de qualité. Il faudrait donc conserver ces différents scénarios.

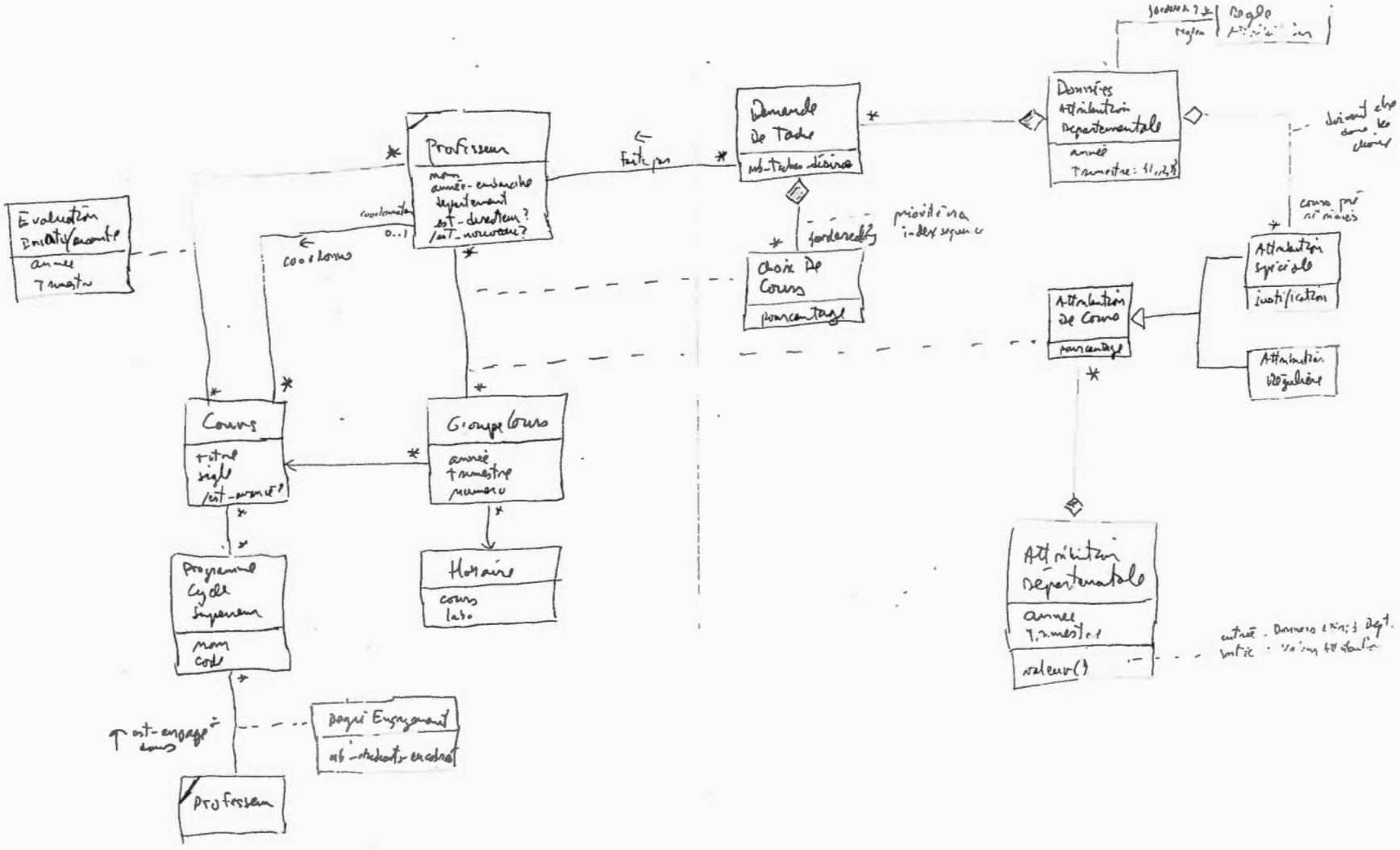
4.7 L'administrateur configure le logiciel

Précondition : Aucune.

	Administrateur		Système
1.	L'administrateur modifie les options de configuration, par exemple, messages d'erreur et courriels à transmettre aux professeurs lorsque les choix contreviennent à des règles, description des règles d'attribution (?).	2.	Vérifie les options.
3.	L'administrateur demande la sauvegarde.	4.	Sauvergarde les nouvelles options.

5 Modèle conceptuel du domaine

Voir page suivante.


Professeur
nom
année-embauche
département
est-directeur?
est-nouveau?
Demande de Tâche
nb-tâches-désirées
Données Attribution Départementale
année
Trimestre: 1,2,3
Evaluation Enseignant(e)
année
Trimestre
Choix De Cours
pourcentage
Attribution de Cours
pourcentage
Attribution spéciale
justification
Attribution régulière
Cours
titre
sigle
est-nouveau?
Groupe Cours
année
trimestre
numéro
Horaire
cours
labo
Programme Cycle Supérieur
nom
code
Attribution départementale
année
Trimestre
valeur()
Projet Encadrement
nb-étudiants-encadrés
Professeur
fait par
coordonne
coordonne
est-engagé dans
priorité ou index séquence
fonderedby
doivent être dans les choix
cours préalables

6 Scénarios client

Cette section présente une série de "scénarios client" — "histoires utilisateur" = *user stories*. Chaque sous-section peut être vue comme approximativement une *itération*. Les scénarios sont présentés sous une forme relativement succincte — à la "Connextra".

6.1 Scénarios de base

S1. Configurer les informations sur les professeurs

*En tant qu'*administrateur

Je veux pouvoir indiquer les professeurs du département et les informations de base qui leur sont associées

Afin que les professeurs puissent ensuite faire leurs choix

S2. Lancer une nouvelle ronde d'attribution

En tant que directeur adjoint à l'enseignement

Je veux lancer l'attribution pour une nouvelle session

Afin de permettre à l'assistante administrative d'indiquer les horaires et aux professeurs d'indiquer leurs choix

S3. Indiquer les horaires de cours

*En tant qu'*assistante administrative

Je veux indiquer les cours disponibles et leurs horaires

Afin de permettre aux professeurs d'indiquer leurs choix et que ces choix soient corrects (cours qui existent vraiment)[1]

S4. Indiquer mes choix de cours

En tant que professeur

Je veux indiquer mon choix de cours

Afin que l'attribution des cours puisse se faire par le directeur adjoint

[1] Cf. remarque de Normand Séguin à ce sujet (courriel du 15 septembre 2013) : "Il ne faut pas que les profs entrent eux-mêmes les sigles et groupes, il y a plein d'erreurs à ce niveau de leur part."

S5. Générer une attribution approximative naïve

En tant que directeur adjoint à l'enseignement

Je veux générer une attribution approximative relativement aux choix indiqués par les professeurs, qui ne tient compte que des règles quant aux conflits d'horaire (algorithme glouton naïf)

Afin d'avoir une attribution à partir de laquelle travailler

6.2 Scénarios intermédiaires

S6. Indiquer les statuts des professeurs

En tant que directeur adjoint à l'enseignement

Je veux pouvoir indiquer le statut de certains professeurs (directeur, sabbatique)

Afin que l'attribution puisse se faire en fonction des règles plus avancées

S7. Modifier/corriger un horaire de cours

*En tant qu'*assistante administrative

Je veux modifier l'horaire d'un cours ou ajouter un nouveau cours à ceux dèjà

Afin que les professeurs aient les bonnes informations et qu'ils puissent, si nécessaire, modifier leurs choix

S8. Spécifier des attributons prédéfinies

En tant que directeur adjoint à l'enseignement

Je veux pouvoir indiquer certaines attributions spéciales, prédéfinies, notamment impliquant des professeurs d'autres départements qui ne sont pas dans la base de données des professeurs

Afin de respecter la règle sur les ententes avec les directions de programme au niveau des cours avancés

S9. Modifier un choix de cours

En tant que professeur

Je veux pouvoir modifier mes choix de cours

Afin de tenir compte des changements d'horaire, des discussions et arrangements que j'aurais pu avoir avec des collègues

S10. Transmettre un rappel aux retardataires

En tant que directeur adjoint à l'enseignement

Je veux pouvoir identifier les professeurs qui n'ont pas encore remis leurs choix

Afin de rappeller aux retardataires qu'ils doivent soumettre leurs choix

S11. Vérifier les choix

En tant que directeur adjoint à l'enseignement

Je veux pouvoir vérifier les choix faits par les professeurs

Afin d'informer certains professeurs, si nécessaire, qu'ils doivent corriger leurs choix

S12. Générer une attribution approximative améliorée

En tant que directeur adjoint à l'enseignement

Je veux générer une attribution approximative relativement aux choix indiqués par les professeurs (par exemple, à l'aide d'un algorithme glouton simple), qui tienne compte de quelques règles

Afin d'avoir une attribution qui tienne compte d'un peu plus de règles

S13. Modifier une attribution et générer une attribution améliorée

En tant que directeur adjoint à l'enseignement

Je veux pouvoir modifier une attribution proposée par le logiciel et, de là, lancer à nouveau le processus de génération

Afin d'avoir une attribution améliorée qui tienne compte de certains choix faits par le directeur adjoint

6.3 Scénarios avancés

S14. Modifier la charge associée à un groupe-cours

En tant que assistante administrative

Je veux pouvoir modifier le pourcentage de tâche associé à un groupe-cours

S15. Configurer les règles d'attribution

*En tant qu'*administrateur

Je veux pouvoir spécifier les règles d'attribution

Afin que l'attribution puisse se faire

S16. Générer une attribution gloutonne

En tant que directeur adjoint à l'enseignement

Je veux générer une attribution gloutonne relativement aux choix indiqués par les professeurs qui tienne compte de toutes les règles

Afin d'avoir une attribution qui tienne compte d'un peu plus de règles

S17. Indiquer les évaluations insatisfaisantes

En tant que directeur adjoint à l'enseignement

Je veux pouvoir indiquer les évaluations insatisfaisantes de certains professeurs

Afin que l'attribution puisse se faire en fonction des règles plus avancées

S18. Indiquer le degré d'engagement des professeurs dans les programmes avancés

En tant que directeur adjoint à l'enseignement

Je veux pouvoir indiquer le degré d'engagement des professeurs dans les programmes d'études avancés (de cycles supérieurs)

Afin que l'attribution puisse se faire en fonction des règles plus avancées

6.4 Scénarios encore plus avancés

S19. Générer une attribution la plus optimale possible

En tant que directeur adjoint à l'enseignement

Je veux générer une attribution relativement aux choix indiqués par les professeurs qui tienne compte de toutes les règles et qui soit la plus optimale possible

Afin d'avoir une meilleure attribution

7 Glossaire

Administrateur	S'occupe de gérer le logiciel. Peut être un professeur, l'assistante administrative, le directeur adjoint à l'enseignement.
Assemblée départementale	Ensemble de tous les professeurs qui sont membres de l'unité d'accréditation SPUQ–UQAM. Exclut le doyen, les vice-recteurs, etc.
Attribution départementale	Attribution, à une session spécifique, de zéro, un ou plusieurs cours *à chacun des professeurs* de l'assemblée départementale.
Coordonnateur	Professeur responsable d'un cours, notamment, de valider le plan de cours, de coordonner les enseignants, etc. Nommé pour une année lors d'une assemblée départementale (généralement en avril ou mai).
Directeur adjoint	Directeur adjoint à l'enseignement, membre de l'exécutif du département. Traditionellement, c'est cette personne qui s'occupe de l'attribution des tâches d'enseignement aux professeurs, à tout le moins qui produit la première ébauche de résolution.
Professeur	Professeur du département d'informatique, ou d'un autre département (parfois, pour certains cours). Pas un chargé de cours, les règles dans ce cas étant définies par la convention collective SCCUQ–UQAM.
Sabbatique	Période (6 ou 12 mois) durant laquelle un professeur n'enseigne pas.

BIBLIOGRAPHIE

(1) Al-Yakoob, S. M., et H. D. Sherali. 2006. « Mathematical programming models and algorithms for a class–faculty assignment problem ». *European Journal of Operational Research*, vol. 173, no. 2, p. 488–507.

(2) Bacchus, F., et P. Van Run. 1995. « Dynamic variable ordering in CSPs ». In *Principles and Practice of Constraint Programming—CP'95*, p. 258–275. Springer.

(3) Blum, C., et A. Roli. 2003. « Metaheuristics in combinatorial optimization : Overview and conceptual comparison ». *ACM Computing Surveys (CSUR)*, vol. 35, no. 3, p. 268–308.

(4) Brailsford, S., C. Potts et B. Smith. 1999. « Constraint satisfaction problems : Algorithms and applications ». *European Journal of Operational Research*, vol. 119, no. 3, p. 557–581.

(5) Carrasco, M., et M. Pato. 2001. *A multiobjective genetic algorithm for the class/teacher timetabling problem.* Coll. « Practice and Theory of Automated Timetabling III », p. 3–17. Springer.

(6) Clausen, J. 1999. « Branch and bound algorithms-principles and examples ». *Department of Computer Science, University of Copenhagen*, p. 1–30.

(7) Erciyes, K. 2014. *Complex Networks : An Algorithmic Perspective.* CRC Press.

(8) Glover, F. 1986. « Future paths for integer programming and links to artificial intelligence ». *Computers & operations research*, vol. 13, no. 5, p. 533–549.

(9) ———. 1989. « Tabu search—part I ». *ORSA Journal on computing*, vol. 1, no. 3, p. 190–206.

(10) ———. 1990. « Tabu search—part II ». *ORSA Journal on computing*, vol. 2, no. 1, p. 4–32.

(11) Goldberg, D. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. T. 412. Addison-wesley Reading Menlo Park.

(12) Gunawan, A., et K. Ng. 2011. « Solving the teacher assignment problem by two metaheuristics ». *International Journal of Information and Management Sciences*, vol. 22, no. 2, p. 73–86.

(13) Gunawan, A., K. Ng et K. Poh. 2007. « Solving the teacher assignment-course scheduling problem by a hybrid algorithm ». *International Journal of Computer, Information, and Systems Science, and Engineering*, vol. 1, no. 2, p. 137–142.

(14) Hmer, A., et M. Mouhoub. 2010. *Teaching assignment problem solver*. Coll. « Trends in Applied Intelligent Systems », p. 298–307. Springer.

(15) Hosny, M. 2012. « A heuristic algorithm for solving the faculty assignment problem ». In *Proceedings of the International Conference on Frontiers in Education : Computer Science and Computer Engineering (FECS)*, p. 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

(16) Kumar, V. 1992. « Algorithms for constraint-satisfaction problems : A survey ». *AI magazine*, vol. 13, no. 1, p. 32–44.

(17) Land, A. H., et A. G. Doig. 1960. « An automatic method of solving discrete programming problems ». *Econometrica : Journal of the Econometric Society*, p. 497–520.

(18) Mackworth, A. 1977. « Consistency in networks of relations ». *Artificial intelligence*, vol. 8, no. 1, p. 99–118.

(19) Russell, S., et P. Norvig. 1995. « Artificial intelligence : A modern approach ».

(20) Tremblay, G. 2014. *Programmation parallèle haute performance — Notes de cours.* <http://www.labunix.uqam.ca/~tremblay/INF7235/Materiel/notes-de-cours.pdf>. Consulté le 2015-09-02.

(21) Vecchi, M. P., et S. Kirkpatrick. 1983. « Global wiring by simulated annealing ». *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 2, no. 4, p. 215–222.

(22) Wikipedia, n. 2015. *Min-conflicts algorithm.* <https://en.wikipedia.org/wiki/Min-conflicts_algorithm>. Consulté le 2015-11-27.

(23) Xia, N. 2006. « A heuristic algorithm for courses assignment ». Mémoire de maîtrise, Montréal, QC, Canada, Université du Québec à Montréal.

(24) Yang, S., et S. N. Jat. 2011. « Genetic algorithms with guided and local search strategies for university course timetabling ». *IEEE Transactions on Systems, Man, and Cybernetics, Part C : Applications and Reviews*, vol. 41, no. 1, p. 93–106.