TER
E-learning Evolutif
Rapport

Maîtrise d'informatique 2003/2004

Etudiants :	**Encadrants :**
Albarelli Corinne Behem Patrice Guillot Jérôme Joly Laetitia Missonier Benoît Rouch Emilie	Pierre Crescenzo Cathy Escazut Michel Gautero Maîtres de conférences en Informatique

Résumé

L'utilisation des NTIC (Nouvelles Technologies de l'Information et des Communications) dans l'enseignement grandit d'année en année. Il existe déjà de nombreux environnements mais ceux-ci ont en général un point faible : contrairement à l'enseignant qui peut parfois s'adapter à l'étudiant, le site Web est inflexible.
L'objectif principal de ce projet est de modéliser le profil d'un étudiant (sa manière de raisonner, sa compréhension d'un cours, sa vitesse de réflexion, …) et de mettre en place les outils permettant de présenter à cet étudiant un cours adapté à son profil. Il y a donc plusieurs aspects :

Déterminer ce qui entre dans le profil d'un étudiant.

Construire les outils permettant de faire évoluer ce profil en fonction du cheminement suivi par l'étudiant.

Construire les outils permettant de présenter un cours différent à chaque étudiant en fonction de son profil.

Sommaire

Introduction

Lors de nos recherches, nous avons trouvé de nombreux sites d'e-learning. En effet cette méthode qui permet aux étudiants d'apprendre tout en restant chez eux intéresse de plus en plus de monde. Cependant ces sites ont dans la plupart des cas un inconvénient majeur : contrairement à l'enseignant qui peut parfois s'adapter à l'étudiant, le site Web est inflexible.
Toute fois il existe des sites d'e-learning qui tentent de s'adapter aux étudiants. C'est le cas de Paraschool, qui est un site d'apprentissage français. Nous nous sommes donc inspirés de ce dernier pour proposer un site qui s'adapte au profil de l'étudiant.

Le point de départ de notre projet est un algorithme (algorithme ACO) qui fut adapté par une équipe de l'INRIA au site d'e-learning de Paraschool, qui présentait à l'origine des cours rigides. En partant de cet algorithme, nous avons tenté de développer un site qui prend en considération les résultats des étudiants précédents ainsi que les préférences de l'étudiant.

Dans un premier temps nous allons rappeler les points essentiels du cahier des charges. Puis nous décrirons en détail le travail réalisé. Ensuite nous évoquerons notre planning et nos méthodes de travail. Enfin nous conclurons sur la difficulté de paramétrer un algorithme et l'expérience acquise grâce ce projet.

1 Cahier des charges

1.1 Fournitures

Nous devons rendre un prototype de système (site + algorithme + autres éléments utiles) qui démontre l'efficacité de l'adaptation.

1.2 Organisation du projet

1.2.1 Processus

Le modèle de développement utilisé est la modélisation en cascade :

- Analyse : étude de l'existant, étude de l'algorithme Ant Colony Optimisation
- Conception : modélisation d'un étudiant, de son profil, modélisation d'un cours,…
- Programmation : programmation de l'algorithme, du site Web, des briques logicielles
- Tests : mise en place de robots étudiants pour tester le bon fonctionnement de l'algorithme

La documentation est effectuée tout au long du projet.

1.2.2 Limites et interfaces

Limites

- Le cours est statique : il est impossible pour un professeur de le modifier et de le faire évoluer.
- Il est nécessaire d'avoir trois versions du même cours (une version entièrement textuelle, une version avec des graphiques, et une dernière version avec des animations).
- Nous avions pensé créer un « éditeur de cours » permettant aux professeurs de proposer un cours de façon plus modulaire. Mais cela est apparu trop fastidieux par rapport au temps qui nous est imparti, et nous avons abandonné cette idée.

Interfaces

La principale interface est celle avec le serveur Apache qui tournera chez le client, et qui gèrera la base de données, grâce au logiciel MySql.

1.3 Gestion

1.3.1 Objectifs et priorités

Objectifs

Nous devons fournir un site Web convivial permettant l'apprentissage personnalisé. Le site doit pouvoir s'adapter à l'étudiant, c'est-à-dire que la présentation et le contenu seront différents d'un étudiant à l'autre. Pour cela, nous allons modéliser le profil d'un étudiant à savoir sa manière de raisonner, sa compréhension d'un cours ou encore sa vitesse de réflexion. Nous mettrons ensuite en place les outils permettant de présenter à chaque étudiant un cours adapté à son profil et de faire évoluer ce profil en fonction du cheminement suivi. Enfin, nous construirons le site Web qui devra être convivial et simple d'utilisation.

Priorités

La tâche prioritaire est la mise en place de l'algorithme par colonies de fourmis (ou ACO pour Ant Colony Optimisation) permettant de faire évoluer un cours en fonction du cheminement de l'étudiant. Ensuite vient la réalisation d'un site convivial.

1.3.2 Hypothèses, dépendances, contraintes

Hypothèses

Le cours est statique, il ne pourra pas être complété ou modifié. Il est constitué de plusieurs modules, chacun correspondant à une page Web. Chaque module du cours existe en trois versions: une version composée uniquement de texte, une de graphiques et une troisième d'animations. Après chaque module, une série d'exercices est proposée à l'étudiant et, en fonction du résultat obtenu, le profil de ce dernier sera modifié.

Lors de la première utilisation du site Web, l'étudiant devra répondre à un questionnaire permettant d'identifier ses préférences. Ces dernières permettront d'établir un premier profil pour cet étudiant.

L'étudiant sera guidé tout au long de son apprentissage. Il choisira simplement un niveau de départ en fonction de ses compétences, puis il sera orienté à travers le site grâce à l'algorithme mis en place. S'il échoue aux exercices de fin de module, son niveau sera revu à la baisse et des modules de niveaux inférieurs lui seront proposés, dans le cas contraire, il pourra passer au niveau suivant.

Contraintes

- La principale contrainte est d'implémenter l'algorithme ACO fourni pour permettre l'évolution du cours.
- Nous devons implémenter des « robots étudiants » afin de simuler différents comportements d'étudiants.

- L’ordinateur serveur doit tourner avec Windows 2000/XP. L’utilisation sous Linux et Mac OS est secondaire.
- L’environnement de développement doit supporter Apache/Php/MySql.

1.3.3 Gestion du risque

La mise en place de l’algorithme ainsi que celle des robots étudiants constituent les deux risques majeurs de ce projet : en ce qui concerne l’algorithme, les risques sont sa mauvaise interprétation et sa mauvaise programmation, menant à la non-évolution du cours. Les robots doivent permettre de gérer au maximum ce risque, en vérifiant que l’étudiant est bien dirigé vers un cours adapté à son profil, et ainsi en contrôlant l’efficacité de l’algorithme.

1.3.4 Moyens de contrôle

- Communication entre les différents acteurs
- Echange régulier d’informations avec les encadrants
- Processus de tests : robots étudiants

1.4 Fonctions du produit

1.4.1 Identification d’un étudiant

- Le système permet à un nouvel étudiant de s'inscrire,
- Le système permet à un étudiant (déjà inscrit) de se connecter.

1.4.2 Le système permet de choisir un cours

- L'étudiant choisit le cours à afficher,
- Le système détermine le cours à afficher.

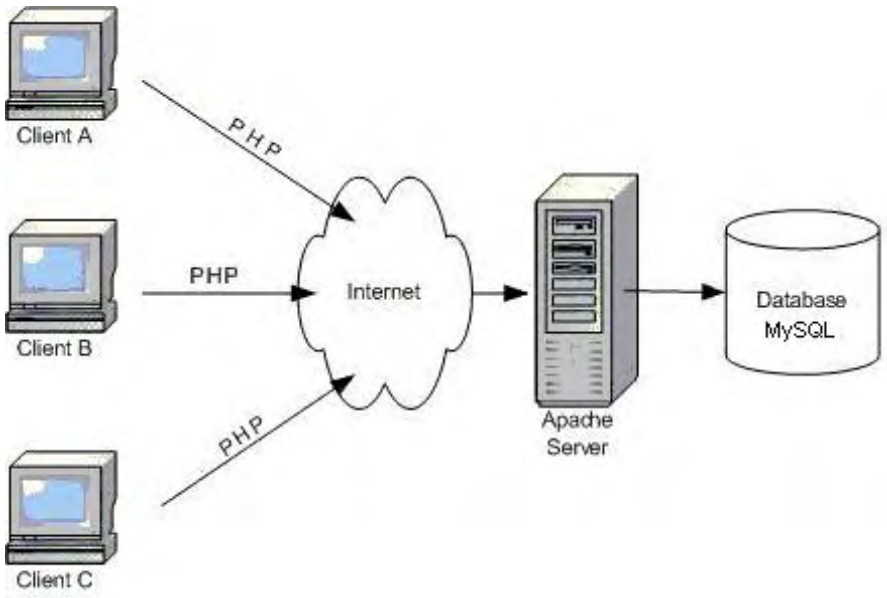
1.4.3 Le système propose des cours et des exercices aux étudiants

- Le système affiche le chapitre d’un cours,
- Le système propose des exercices après chaque chapitre,
- Le système fait avancer l'étudiant à la page suivante.

1.4.4 Evolution des différents paramètres

- Le système choisit un niveau de départ pour un cours et un étudiant
- Le système peut faire évoluer le niveau d'un étudiant,
- Le système peut faire évoluer le niveau d'un exercice,
- Le système permet le changement des préférences.

1.5 Architecture logicielle



2 Descriptif du travail réalisé

2.1 Introduction

Le but de notre projet est la création d'un site d'e-learning s'adaptant au profil de chaque étudiant. Les cours présentés doivent permettre d'optimiser l'apprentissage de chacun. Pour ce faire, il nous a été demandé d'utiliser puis d'adapter l'algorithme d'optimisation par colonies de fourmis et de l'appliquer sur notre site.

Nous avons donc dû dans un premier temps comprendre le fonctionnement de cet algorithme. Nous avons dû également déterminer les critères entrant dans la définition du profil de l'étudiant, et ce qu'était un bon cours pour tous les étudiants, quel que soit leur profil. Il a fallu alors trouver une interface permettant à l'étudiant d'obtenir un cours adapté à son profil. Nous avons pu ensuite aborder la phase d'implémentation, pour finir par une phase de test.

Nous avions dans un premier temps penser réaliser des briques logicielles afin de permettre l'évolution du profil de l'étudiant. Mais nous avons changé leur nom en « évaluateur », puisque nous ne les avons pas implémentées comme des bouts de code réutilisables.

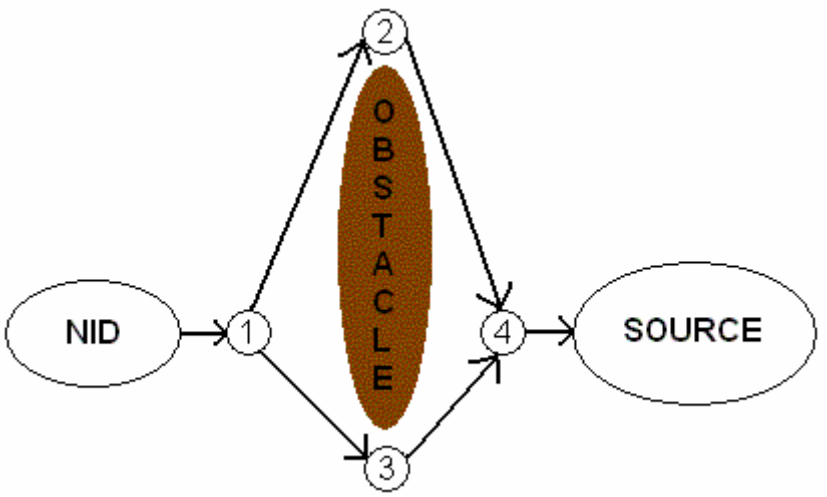
2.2 L'algorithme par colonies de fourmis

Des articles nous ont été fourni lors de notre première réunion avec nos encadrants. Ces articles ont été écrits à la suite d'une collaboration entre l'INRIA et la société Paraschool. Ils montrent comment appliquer l'algorithme d'optimisation par colonies de fourmis (ou ACO : Ant Colony Optimisation) dans le cadre de sites d'e-learning. Ils ont été à la base de la réalisation de notre TER. C'est pourquoi nous allons citer des passages de ces articles pour expliquer cet algorithme ainsi que son application à un site d'e-learning.

Dans cette partie, tous les paragraphes en italique sont des citations extraites de l'article « Application de l'optimisation par colonies de fourmis à la structuration automatique de parcours pédagogiques » de Yann Semet et Pierre Collet.

Explication de l'algorithme d'optimisation par colonies de fourmis : ACO

Pour mieux comprendre la façon dont fonctionnent les colonies de fourmis, reprenons l'exemple traditionnellement donné pour illustrer leur capacité à trouver des chemins optimaux. La figure ci-dessous illustre une situation où il y a un nid, où les fourmis vivent, et une source de nourriture, que les fourmis doivent trouver et dont elles doivent ramener les provisions vers le nid.



Un problème naturel typique : un nid, une source de nourriture et deux chemins, un court, un long.

Il existe deux chemins possibles pour atteindre la source : un long, un court. Les fourmis explorent aléatoirement les deux. Lorsqu'elles trouvent la source, elles se chargent de nourriture, et retournent au nid par où elles sont venues en libérant des phéromones tout au long de leur chemin. Le chemin le plus court étant aussi le plus rapide, sa concentration en phéromones augmentera plus vite et les fourmis, qui suivent les traces chimiques, seront très vite, par un phénomène de renforcement, encouragées à suivre le chemin le plus court. Le chemin du nid à la source a été optimisé. Cette façon de procéder est déjà efficace en soi mais il manque deux phénomènes essentiels pour qu'elle soit tout à fait complète.
Le premier phénomène est qu'il arrive quelquefois que des fourmis étourdies se trompent et s'écartent du chemin de phéromones. Si, par chance, une fourmi égarée par erreur trouve un chemin plus court, la trace de phéromone qu'elle laissera derrière elle sera plus fraîche, indiquant par là même aux autres fourmis qu'il existe un chemin plus court pour accéder à la nourriture. Ainsi, c'est le mécanisme d'erreur dans le suivi de trace de phéromone qui permet la découverte de raccourcis, aboutissant à l'établissement d'un chemin optimal entre fourmilière et nourriture.
Le deuxième phénomène est que les phéromones s'évaporent dans le temps, rendant ainsi leurs traces éphémères. C'est ce deuxième mécanisme qui permet aux chemins établis de ne pas être statiques, et de s'adapter aux modifications de l'environnement. Supposons maintenant que seul le chemin le plus long soit disponible (une brindille obstrue le chemin court). Les fourmis vont donc l'utiliser et le charger en phéromones. Supposons maintenant que le chemin redevienne tout à coup disponible (la brindille a été poussée par le vent), notre colonie se trouve dans une situation non optimale : une piste de phéromones encourage les fourmis à suivre un chemin qui n'est pas le meilleur puisqu'elles suivent le chemin long alors que le court est à nouveau disponible. La situation est rétablie par le caractère volatil des phéromones : sans apport de phéromone important, le chemin le plus long finit par s'effacer pour disparaître presque complètement. Grâce à l'action conjuguée de l'évaporation et de la croissance rapide de la concentration en phéromones sur le chemin court, le chemin long va rapidement tomber en désuétude et l'optimalité sera rétablie.

Application à un graphe

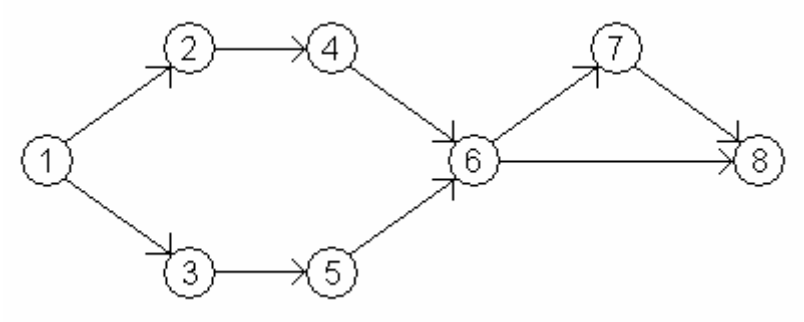
L'algorithme ACO se transpose aisément sur un graphe : les arcs correspondent aux chemins que peuvent suivre les fourmis, et les nœuds aux points de rencontre de ces différents chemins.

A chaque nœud, le choix d'un chemin par une fourmi se fait au hasard en fonction d'une probabilité. Cette probabilité peut être donnée par une formule qui tient compte des taux de réussite et d'échec des fourmis précédemment passées par ces liens. Plus le nombre de passage sera important, plus on va voir se dessiner un chemin optimal selon les critères que l'on aura défini.

Du graphe au e-learning

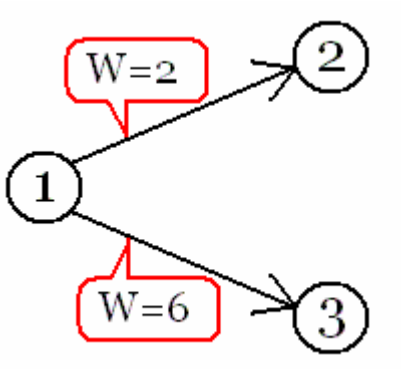
Pour appliquer l'algorithme ACO à un cours, on peut donc représenter celui-ci sous la forme d'un graphe, et même plus précisément sous la forme d'un arbre.
Les noeuds sont des modules du cours (chapitre, page d'exercices, etc.), et les arcs les liens qui permettent de passer d'un module à l'autre. Les liens hypertextes d'un site Internet se prêtent particulièrement à cette représentation.

Voici un exemple de représentation possible d'un cours :



La valeur W : le coefficient pédagogique

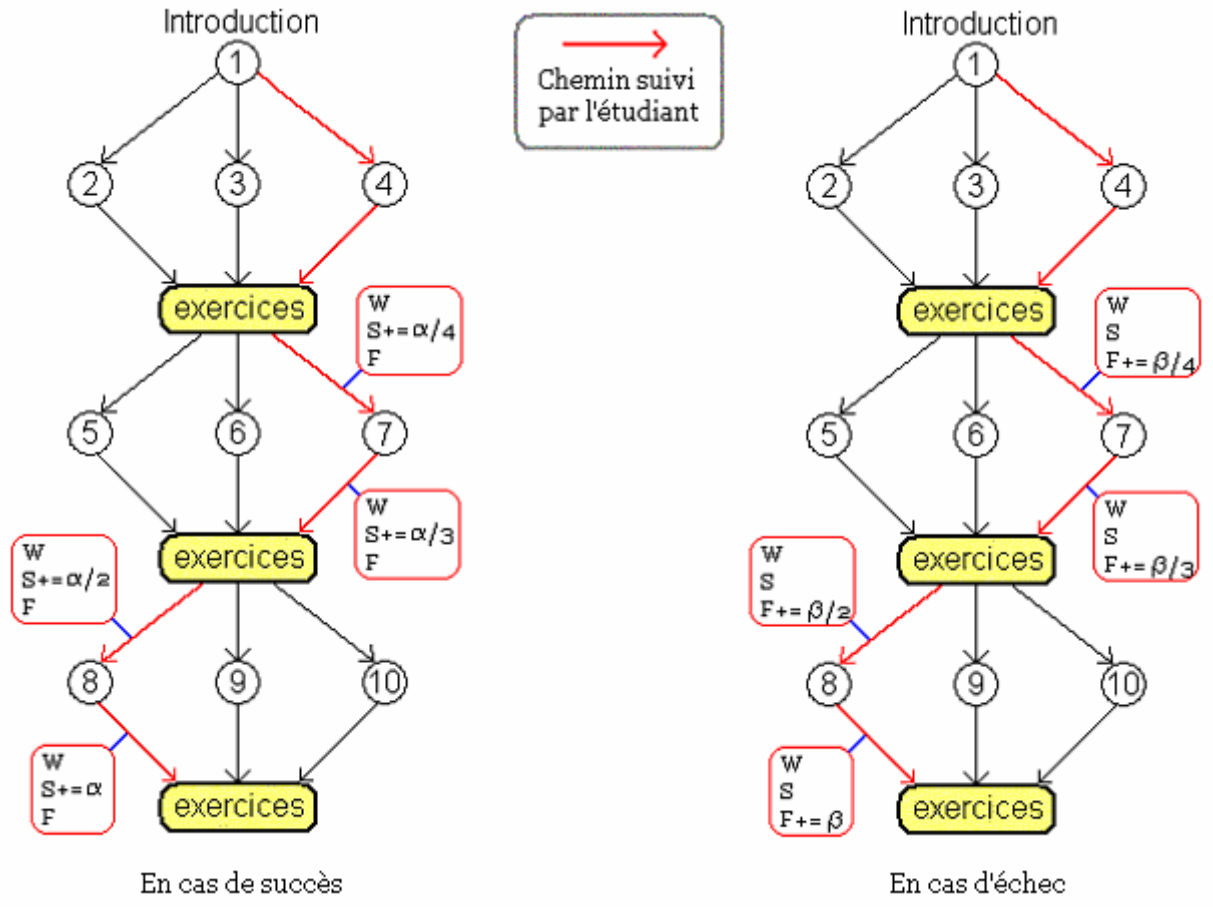
Sur chaque arc nous définissons une valeur W qui représente la préférence accordée par le professeur à un lien. Plus W est élevé, plus ce lien est conseillé par le professeur aux étudiants. Bien sûr l'importance de ce coefficient pour un lien donné est toute relative car elle dépend entièrement de la valeur des coefficients des autres liens qui partent du même nœud.



Les valeurs S et F : les phéromones

*Chaque élève qui parcourt le graphe est représenté par une « fourmi », un micro agent, qui navigue sur le graphe sous-jacent. À l'issue de chaque exercice, ou de chaque leçon suivie d'un questionnaire, la fourmi libère des phéromones. Si l'exercice est validé avec succès, ce seront des phéromones de succès (**S**), sinon ce seront des phéromones d'échec (**F** pour Failure). Chaque arc portera donc, en plus de **W**, deux valeurs **S** et **F**.*
Rétro-propagation et portée pédagogique spatiale

*Les phéromones, qu'il s'agisse de **S** ou de **F** ne sont pas simplement libérées sur l'arc qui a mené la fourmi au noeud/exercice courant mais sur les n derniers arcs que la fourmi a suivi. Ceci est fait pour refléter le fait pédagogique que le succès (ou l'échec) à un endroit donné est conditionné par ce qui a été vu avant par l'élève. Bien évidemment, cette influence diminue avec le temps et l'espace : plus le noeud est éloigné dans l'histoire de la fourmi, moins il a d'importance. Pour que cela soit pris en compte, la quantité de phéromone déposée diminue à mesure que la rétro-propagation avance. Numériquement, n est fixée à 4 et l'amplitude diminue en 1/k partant d'une valeur α prédéfinie, paramètre du système. Les figures 10 et 11 illustrent la rétro-propagation : après avoir réussi (ou échoué) au noeud 7, une quantité α de phéromones est déposée sur l'arc 6->7, une quantité $\alpha/2$ sur l'arc 4->6, $\alpha/3$ pour 2->4 et $\alpha/4$ pour 1->2.*



Rétro-propagation des phéromones de succès et d'échec

Evaporation

Le fait que les phéromones naturelles s'évaporent avec le temps est extrêmement important car cela permet à la colonie de fourmis de se fier à des informations constamment mises à jour. Dans notre système artificiel, il est important d'implémenter une forme d'évaporation pour éviter que le système ne reste « coincé » dans un optimum local ainsi que pour ouvrir la porte aux caractéristiques attendues d'adaptabilité dynamique.

$$S_t = \tau^x S_{t-1}$$

*L'équation donne la forme de l'évaporation pour les phéromones de succès **S** (l'équation est identique pour **F**) : τ, le taux d'évaporation, est un paramètre clé du système dont on verra l'une des conséquences dans les tests de simulation décrits plus bas. La période d'évaporation, x qui dit à quels intervalles l'évaporation est calculée est une constante pédagogique. Typiquement, τ=0.999 et x=1 jour.*

La fitness

C'est notre algorithme qui doit déterminer le chemin à emprunter par l'étudiant. Lorsque l'étudiant veut changer de nœud, nous devons déterminer quel lien il doit parcourir. Pour ce faire nous attribuons à chaque lien une valeur, appelée fitness. Une fois les fitness calculées, on choisit au hasard un lien selon des probabilités calculées à partir des fitness.
Le calcul de la fitness pour un étudiant e et un lien l est donné par les valeurs définies précédemment (W, S et F), ainsi que par les coefficients personnels que l'on peut rajouter (P(e , l)) :

$$f(e,l) = \mathrm{P}(e,l)(\omega_W W + \omega_S S - \omega_F F)$$

Les valeurs ω_W, ω_S, et ω_F sont des pondérateurs. Ils permettent de donner plus ou moins d'importance à chacun des coefficients, paramétrant ainsi la réactivité de l'algorithme.
La fitness sera donc d'autant plus élevée que le lien est encouragé par le professeur (W élevé), que les succès ont été nombreux (S), que les échecs ont été peu élevés (F), et que la préférence personnelle de l'étudiant sur ce lien est importante (P(e , l)).

Les algorithmes de sélections

A partir d'un nœud, le prochain lien à suivre est choisi en fonction de la valeur des fitness des différents liens partant de ce noeud.

Il existe plusieurs modes de sélections, tous reposant sur des probabilités. Nous allons en décrire quelques-uns.

Sélection par roulette

La probabilité de sélectionner chaque arc est strictement proportionnelle à sa fitness ; cette méthode est entièrement automatique, il n'existe aucun moyen de régler la pression sélective. L'avantage, c'est que cette méthode est sensible eux variations subtiles dans les mesures de fitness mais il y a un revers à cette médaille de sensibilité : si d'aventure un arc devient prééminent au point d'en écraser les autres, ces derniers n'auront pratiquement plus aucune chance d'être suivis et l'exploration disparaîtra au profit d'une exploitation entêtée.

$$p(a_{n_i n_j}) = \frac{f(a_{n_i n_j})}{\sum_{n_k \in E} f(a_{n_i n_k})}$$

Sélection par rang, seuil manuel

Pour pallier ce défaut, on peut utiliser une sélection par le rang : la probabilité d'être choisi est inversement proportionnelle au rang. S'il y a trois arcs par exemple, le meilleur d'entre eux aura une probabilité 3/(1+2+3) = 3/6 = 1/2 d'être choisi, le second une probabilité de 2/6 = 1/3 et le dernier une probabilité de 1/6. Un problème est évité mais au détriment de la sensibilité aux variations graduelles. Qui plus est, on n'a pas non plus de contrôle sur la pression sélective.

Sélection par le rang, seuils manuels

Plutôt que de calculer automatiquement les probabilités de choix en fonction du rang, on attribue manuellement une probabilité à chaque rang. On pourra décider par exemple que le premier a une probabilité 0.77 d'être choisi, le second 0.13, le troisième 0.05 et le quatrième également. Il s'agit bien d'une sélection par le rang, avec son avantage et son inconvénient mais elle est complètement paramétrable, au prix d'un réglage sans doute fastidieux des différents seuils, vu qu'il existe des milliers d'arcs dans le graphe.

Sélection par tournoi

Cette procédure très classique consiste à tirer s individus au hasard et de choisir le meilleur. On voit bien que s conditionne la pression sélective : plus s est grande, plus il est probable que les arcs forts l'emportent. Outre son efficacité algorithmique, cette méthode a l'avantage d'être très simplement paramétrable.

Le choix de l'algorithme de sélection est très important pour donner à l'algorithme ACO le comportement souhaité avec une réactivité plus ou moins importante. Nous avons choisi d'implémenter les quatre algorithmes cités ci-dessus.

2.3 L'Etudiant : un profil

Définition du profil de l'étudiant

Pour proposer des cheminements de cours différents, nous avons cherché des critères permettant de différencier les étudiants dans leur apprentissage.
Nous avons alors défini le profil d'un étudiant : il s'agit de caractéristiques personnelles qui permettent de retrouver dans les cours ce qui convient le mieux à cet étudiant.
Dans un premier temps, nous avions imaginé trois points qui nous semblaient réalistes et réalisables :

- Une préférence de présentation (uniquement du texte, du texte et des illustrations, ou du texte et des animations)
- Une préférence selon l'aspect théorique ou pratique
- Le Niveau de l'étudiant

Mais nous avons constaté qu'il fallait tester notre projet avec un minimum de paramètres afin de s'assurer de son bon fonctionnement. Non pas que trop de paramètres feraient « planter » notre algorithme, mais nous avons souhaité nous focaliser sur un seul avant d'en rajouter d'autres. Nous n'avons donc conservé que la préférence sur la présentation des cours, tout en gardant à l'esprit qu'une évolution ultérieure serait toujours envisageable.

Attribution d'un profil à un étudiant

Dans un premier temps, l'étudiant détermine son profil en répondant à un questionnaire. Mais ce profil peut évoluer si l'étudiant réussit mieux sur des pages de cours qui ne correspondent pas à son profil. Ce changement doit se faire de manière automatique et transparente pour l'étudiant. En vue d'une éventuelle utilisation réelle de notre site, nous laissons également la possibilité à l'étudiant de changer son profil à tout instant.

A partir de son profil, nous devons présenter à l'étudiant les pages de cours correspondantes. Il faut donc définir exactement ce qu'est un cours.

2.4 Les cours

Les caractéristiques des cours

La caractéristique utilisée est la différence de présentation des pages de cours. Nous avons choisi trois présentations possibles pour notre projet :

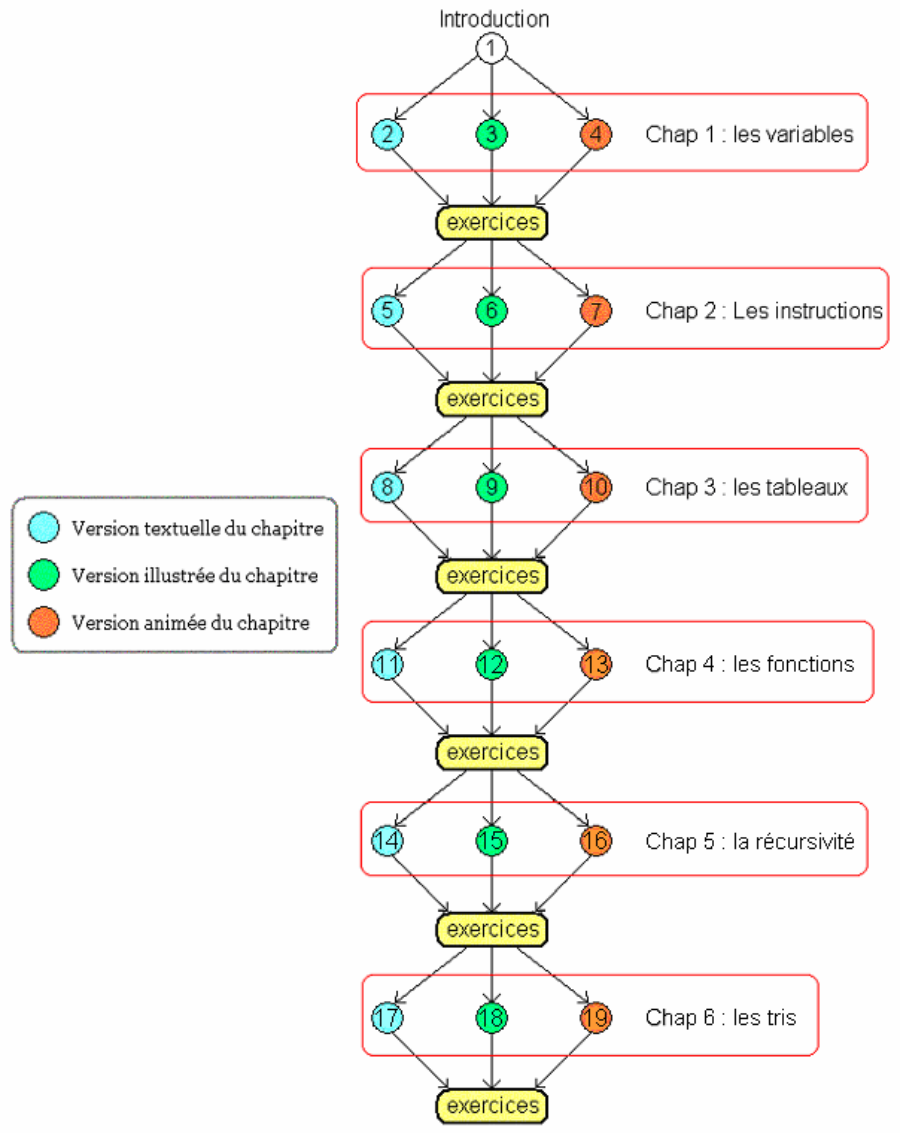
1. Uniquement du texte
2. Texte et illustrations
3. Texte et animations

Pour chaque page de cours, il faut donc prévoir trois pages correspondant à chacune des formats.

La représentation des cours

Comme nous l'avons vu précédemment, les cours sont représentés sous forme d'arbre. Les nœuds modélisent les pages de cours, et les arcs représentent les liens entre ces pages.

Voici la représentation du cours utilisé pour notre projet :



Toute forme d'arbre est possible, mais nous avons choisi une version simplifiée.

La création des cours

La représentation en arbre ainsi que l'attribution de caractéristiques rendent la création de cours assez fastidieuse. En effet, le professeur doit définir les pages, leurs donner des caractéristiques (ici la présentation) et enfin les relier par des liens. On peut donc imaginer que les cours doivent répondre à un protocole qu'il faut définir. Pour une utilisation professionnelle, la création d'un éditeur de cours semble indispensable. Nous ne l'avons pas réalisé car cela ne rentrait pas dans le cadre de notre TER.

L'interface

L'étudiant n'a pas à choisir la page suivante qu'il va consulter : pour pouvoir y accéder, il doit cliquer sur un bouton « SUIVANT » présent dans chaque page de cours. C'est

l'algorithme qui va déterminer la prochaine page à visiter, en fonction de la page courante ainsi que de nombreux paramètres (notamment le profil).

Nous allons maintenant exposer le fonctionnement de notre algorithme.

2.5 De l'algorithme des fourmis au nôtre

L'évolution que nous devons apporter à l'algorithme est l'intégration et le calcul des paramètres personnels. Nous devons aussi nous assurer de leur cohérence et de leur bon fonctionnement, et essayer de trouver le comportement de l'algorithme une fois leur mise en application réalisée.

Intégration des paramètres personnels

Le point le plus important de l'algorithme repose sur le calcul de la fitness d'un arc. Nous rappelons la formule vue précédemment qui la calcule.

$$f(e,l) = \mathrm{P}(e,l)(\omega_W W + \omega_S S - \omega_F F)$$

Lorsque l'étudiant change de page, nous calculons alors les fitness de chaque lien partant de la page où il se trouve. Puis, en fonction des résultats obtenus, l'algorithme de sélection en choisi un pour l'étudiant. Nous voyons que ce calcul prend en compte un paramètre propre à l'étudiant, ici P(e, l). Dans les différents articles qui nous ont été fournis, il ne nous a pas été donné de détail quant à l'intégration des paramètres personnels.
Nous avons donc produit la formule suivante en nous inspirant de la précédente :

$$f(e,l) = (\beta_1 P_1 + \ldots + \beta_n P_n) * (\omega_W W + \omega_S S - \omega_F F)$$

avec $P_1 \ldots P_n$ les valeurs des paramètres personnels et β_1 … β_n leurs pondérateurs respectifs.
L'intérêt de cette formule réside dans l'addition des paramètres personnels qui nous permettent grâce à leurs pondérateurs de leur donner plus ou moins d'importance, ainsi que la multiplication de ceux-ci aux paramètres collectifs qui entraîne une pondération de la fitness collective par un critère personnel.

Comment modifier les paramètres collectifs

Le paramètre W est défini par le professeur pour chaque lien.
Les paramètres S et F sont modifiés lors de la réussite ou de l'échec d'un exercice.

Comment trouver les paramètres personnels

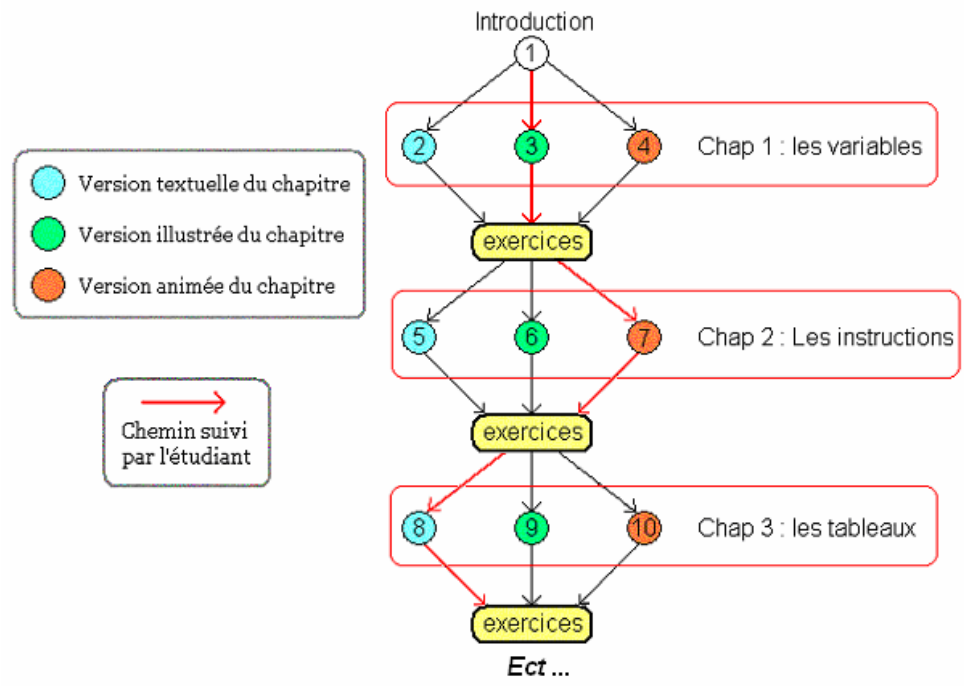
Pour trouver les $P_1 \dots P_n$ nous avons défini un nouveau concept : les évaluateurs.
Chaque évaluateur a pour but de fournir à l'algorithme une valeur P_i en fonction d'un étudiant (de son profil) et d'un lien.
Les évaluateurs sont développés de manière indépendante à l'algorithme, et les valeurs qu'ils renvoient peuvent être contrôlées par les pondérateurs β_i.
Le nombre d'évaluateurs peut varier et ils s'intègrent parfaitement à l'algorithme. Cela signifie une possibilité d'extension des paramètres personnels, c'est-à-dire du profil de l'étudiant et donc de notre projet.

Les évaluateurs

Ce sont des fonctions qui collectent, dans la table *infoetudiant*, différentes informations sur les étudiants lors de leur apprentissage. Ces informations nous permettent par la suite de modifier les préférences de l'étudiant. En effet, l'étudiant choisit au départ la présentation du cours qu'il estime la plus adaptée à sa compréhension. Cependant lors de son cheminement, on peut se rendre compte que la présentation qu'il a choisie (ou qu'on lui a attribuée) ne s'avère pas être la plus adéquate et il faut donc la modifier.
Dans notre projet, la seule préférence de l'étudiant que nous prenons en compte est la présentation d'un cours.

Le comportement de l'algorithme : collectif contre individuel

L'algorithme ACO n'est basé que sur des critères collectifs. Une fois lancé, l'algorithme nous donne un chemin optimal pour l'ensemble des participants. L'intégration des paramètres personnels doit entraîner l'apparition d'un chemin optimal par profil.



Le schéma ci-dessus représente un exemple de ce que l'on espère obtenir. On remarque que ce chemin ne parcourt pas que les liens qui correspondent à ce profil. Certaines pages peuvent être considérées comme inefficaces même si leur profil correspond à celui de l'étudiant. Ceci est dû au calcul de la partie collective de la fitness des liens qui prend en compte les réussites et les échecs obtenus.

Le comportement de l'algorithme : réaliste ?

L'algorithme est réaliste si l'on obtient bien les résultats décris ci-dessus avec de vrais étudiants. Mais son comportement dépend avant tout de leur façon de réagir. Ainsi, un étudiant a-t il vraiment une préférence de présentation pour les cours ? Le temps passé sur la page est-il vraiment un facteur réaliste pour déterminer un comportement humain ? Nous avons trouvé ces facteurs de manière subjective. Mais en cas d'utilisation avérée de l'application, il faudrait probablement les redéfinir ou au moins les affiner, mais il est difficile de prévoir un comportement humain.

Cependant pour tester l'algorithme ces facteurs nous suffisent. En effet, nous l'avons testé avec des « robots étudiants » dont nous avons défini le comportement en fonction des facteurs que nous avons choisi.

Modification du profil de l'étudiant

Lors de l'implémentation de l'algorithme ACO, nous avons également pris le parti de favoriser les résultats obtenus par cet algorithme par rapport aux souhaits de l'étudiant. En d'autres termes, si au fur et à mesure de l'avancement d'un étudiant, les résultats enregistrés par l'algorithme permettent de déduire que cet étudiant s'en sort mieux avec un type de présentation différent de celui qu'il a lui-même choisi, alors l'algorithme prend l'initiative de modifier la préférence de l'étudiant dans la base de donnée.

De manière précise, nous avons décidé de changer la préférence de l'étudiant lorsque la partie personnelle de la fitness d'un lien qui ne correspond pas au profil qu'il a choisi est k fois supérieure à celle de la fitness qui correspond à ce profil.

Pour être pertinent, ce changement automatique de préférence ne peut avoir lieu qu'après un certain temps d'utilisation par l'étudiant; nous avons dû en tenir compte.

Une phase d'apprentissage

Il est à noter que l'algorithme ne peut déterminer ces chemins optimaux qu'après un grand nombre de passages d'étudiants. Même si les critères individuels servent à aiguiller l'étudiant sur un des nœuds accessibles en fonction de son profil, le critère collectif permet quant à lui de l'aiguiller en fonction de l'efficacité de ces nœuds. Le bon fonctionnement de l'algorithme dépend de l'utilisation de ces deux critères simultanément.
Pour remplir les critères collectifs il faut que le cours soit parcouru par de nombreux étudiants : c'est la phase d'apprentissage.

Les tests et le paramétrage

Le calcul des fitness dépend de nombreux paramètres. Nous ne pouvons connaître à l'avance les valeurs des paramètres qu'il faut donner pour que l'algorithme ait le comportement souhaité. Nous devons donc prévoir une phase de test qui nous permet de le paramétrer correctement.

2.6 Passage à l'implémentation

Cette partie a été réalisée par Patrice Behem et Jérôme Guillot, pendant une durée de 15jours à temps plein, auxquels doivent s'ajouter plusieurs journées de travail ultérieures pour effectuer quelques améliorations.

Nous décrivons ici les principales phases de l'implémentation, ainsi que les problèmes rencontrés.

La représentation du cours

La première phase du travail à réaliser a été la construction d'un cours en tenant compte des trois présentations possibles (texte, illustration, animation) et en l'intégrant dans la base de données sous forme d'arbre.

Implémentation de l'algorithme collectif

Une fois le cours et sa représentation achevés, nous avons implémenté l'algorithme ACO, c'est-à-dire uniquement la partie collective de l'algorithme final. Dans ce cas la fitness des liens est donnée par cette formule :

$$f(l) = \omega_W W + \omega_S S - \omega_F F$$

Nous avons stocké les informations collectives au niveau des liens puisqu'elles sont partagées par tous. Ces informations se limitent aux valeurs de W, S et F.

Les pondérateurs ω_1, ω_2, et ω_3 ont été placés dans la table **variable**.

L'algorithme que nous avons programmé dépend d'un grand nombre de paramètres. Pour faciliter la phase de paramétrage ultérieure, nous avons regroupé tous ces paramètres dans la table « variable ».

Au départ, le calcul des fitness ne prenait en compte que W. S et F étaient mis à 0. Notre algorithme de sélection des nœuds était la « roulette ».
Nos premiers tests nous ont montré qu'il est très difficile de tester l'algorithme au fur et à mesure de son avancement. En effet, le choix du lien à parcourir repose sur deux choses : le calcul de la fitness de tous les liens possibles qui partent du même nœud, et le choix d'un de ces liens par probabilité. A cet instant la fitness ne dépendait que de W, il était donc facile de

tester le bon fonctionnement. Mais lorsque le calcul de la fitness dépend de nombreux paramètres, contrôler la fiabilité de l'algorithme est peu aisé.

Les valeurs des coefficients S et F sont modifiées lorsqu'un étudiant passe par un exercice. S'il réussit ou s'il échoue cet exercice, ce sont respectivement les valeurs du coefficient S ou F des arcs qu'il a visités qui sont incrémentées.

Les tests effectués nous ont montré l'importance du paramétrage de l'algorithme : durant toute la phase d'implémentation, nous avons travaillé avec des valeurs choisies arbitrairement. Pour tester valablement l'algorithme et tirer des conclusions sur la valeur des paramètres, un grand nombre de passages est nécessaire. Il est donc difficile de savoir si l'algorithme a le comportement souhaité à chaque phase d'implémentation réalisée.

Des cas particuliers

Les premiers tests de l'algorithme effectués ont également mis en évidence deux cas particuliers principaux d'exécution, ces cas particuliers pouvant être plus ou moins gênants pour la pertinence des choix faits par l'algorithme.

Le premier cas particulier concerne la fitness des liens. En effet, il peut arriver au cours de l'exécution de l'algorithme ACO, que le calcul d'une fitness renvoie un nombre négatif ou nul. Dans les deux cas, le calcul de la probabilité que ce lien soit tiré est totalement faussé. Nous avons alors décidé de fixer une valeur minimale pour les fitness. Ainsi toute fitness est assurée de ne pas descendre en deçà de cette valeur, et assure de fait la justesse des probabilités prises en compte par l'algorithme.

Le second cas particulier est en fait un cas particulier de rétro-propagation des échecs et des succès lors d'un retour en arrière dans le cours déclenché par un échec à une page d'exercice. Nous nous sommes effectivement très vite rendu compte que dans ce cas précis, lors du deuxième passage sur la page d'exercice (précédemment échouée), la réussite ou un nouvel échec ne devait avoir de conséquence que sur la dernière page visitée. Il n'y a donc pas dans ce cas-là de réelle rétro-propagation sur plusieurs nœuds précédents.

Les exercices

Cette partie a été réalisée en binôme par Benoît Missonier et Emilie Rouch pendant 11 jours à plein temps. La rédaction du rapport concernant cette partie a été réalisée en parallèle.

Contrairement aux pages de cours, les pages d'exercices ne sont pas fixes. En effet, pour éviter qu'un étudiant ne tombe toujours sur les mêmes exercices, ces derniers sont choisis de manière aléatoire.

<u>Caractéristique d'un exercice</u>

Un exercice est caractérisé par un niveau de difficulté affecté au départ par l'enseignant. Celui-ci doit proposer plusieurs exercices de chaque niveau de difficulté afin qu'un étudiant, ayant échoué, ait peu de chance lors de sa prochaine évaluation de retomber sur les mêmes exercices. Il faut impérativement que l'enseignant propose au moins autant d'exercices qu'il y a de niveaux de difficultés.

Tirage au sort des exercices

La page d'exercices est composée de cinq exercices. Il existe cinq niveaux de difficulté, et normalement un exercice pour chacun de ces niveaux. Mais ces derniers peuvent évoluer au cours du temps. Il s'agit donc d'établir un algorithme qui prenne en compte cette considération et le gère au mieux. Ainsi, pour chaque niveau de difficulté, un exercice est tiré au sort. Cependant si aucun exercice présent dans la base de données ne correspond au niveau demandé, alors on présente un exercice du niveau supérieur. Une page est donc composée d'autant d'exercices qu'il y a de niveaux de difficulté. Ces exercices sont présentés en ordre croissant sur leur niveau de difficulté.

Evolution du niveau de difficulté d'un exercice

Le niveau de difficulté d'un exercice peut évoluer au cours du temps selon son taux de réussite. Si beaucoup d'étudiants échouent sur un exercice, cela signifie que l'exercice proposé est assez difficile et son niveau peut par conséquent être revu à la hausse. Inversement si les étudiants réussissent bien un exercice son niveau peut être revu à la baisse. Voici comment nous avons géré ce fait :

$$\textit{Taux de réussite d'un exercice} = \frac{(\textit{Nombre de fois qu'un exercice est réussi})}{(\textit{Nombre de fois qu'un exercice est proposé})}$$

Les champs « *nbReussi* » et « *nbPropose* », conservés dans la table **exercice** de la base de données, correspondent respectivement au dividende et au diviseur de la formule ci-dessus.

Nous avons décidé de proposer cinq niveaux de difficulté par page d'exercices (n = 5).

Ainsi, le niveau d'un exercice change selon la table d'inéquations suivante :

$$\frac{n-1}{n} \leq \text{Taux de réussite} \rightarrow \text{niveau de l'exercice} = 1$$

$$\frac{n-2}{n} \leq \text{Taux de réussite} < \frac{n-1}{n} \rightarrow \text{niveau de l'exercice} = 2$$

$$\frac{n-3}{n} \leq \text{Taux de réussite} < \frac{n-2}{n} \rightarrow \text{niveau de l'exercice} = 3$$

….

$$\frac{1}{n} \leq \text{Taux de réussite} < \frac{2}{n} \rightarrow \text{niveau de l'exercice} = n-1$$

$$\text{Taux de réussite} < \frac{1}{n} \rightarrow \text{niveau de l'exercice} = n$$

Il est évident que nous ne pouvons pas mettre à jour le niveau de difficulté d'un exercice chaque fois que celui-ci est présenté à un étudiant. En effet, l'initialisation du niveau de difficulté des exercices effectuée par l'enseignant serait alors inutile puisque lorsqu'un exercice est proposé une première fois, soit l'étudiant réussit et le niveau de difficulté devient 1, soit il échoue et le niveau de difficulté devient n. Nous avons donc choisi de le mettre à jour chaque fois qu'un exercice est proposé à 20 étudiants.

Implémentation

Tous les exercices présentés aux étudiants sont des QCM dont le numéro de la réponse est conservé dans la table *exercice* de la base de données.
Un exercice est en fait un morceau de code *php* qui contient l'énoncé de la question et les réponses proposées avec les boutons pour sélectionner la réponse.

Exemple : exercice correspondant au chapitre sur les fonctions.

```
<?
echo "<br><h3>Exercice (Fonctions a) :</h3>\n";
echo "<b>Question :</b> énoncé de la question (difficulté initiale de l'exercice :
1).<br>";
echo "<input type='radio' name='ExoFonc1a' value='1'>Vrai\n";
echo "<input type='radio' name='ExoFonc1a' value='2'>Faux\n";
?>
```

La page d'exercices est la même pour tous les chapitres du cours car on inclut le code des exercices dans cette page. Les exercices à inclure sont sélectionnés par la méthode vue précédemment.
A partir de l'*idPage* du chapitre auquel la page d'exercices se rapporte, on peut choisir les exercices qui lui correspondent dans la base de données.

Insertion de paramètres individuels

Une fois l'algorithme collectif terminé, nous devons y insérer les paramètres personnels.

$$f(e,l) = (\beta_1 P_1 + \ldots + \beta_n P_n) * Partiecollective$$

Le calcul des paramètres personnels est effectué par les différents évaluateurs. Il a donc fallu trouver un moyen d'intégrer aux calculs de l'algorithme les valeurs renvoyées par ces évaluateurs.
Comme les paramètres personnels sont définis de manières subjectives, on devrait pouvoir en rajouter un nombre important. Mais chaque rajout d'un paramètre conduit obligatoirement à un nouveau paramétrage de l'algorithme si on souhaite avoir un comportement cohérent et contrôlé.

Les évaluateurs

Cette partie a été réalisée en binôme par Benoît Missonier et Emilie Rouch pendant 9 jours à plein temps. La rédaction du rapport concernant cette partie a été réalisée en parallèle.

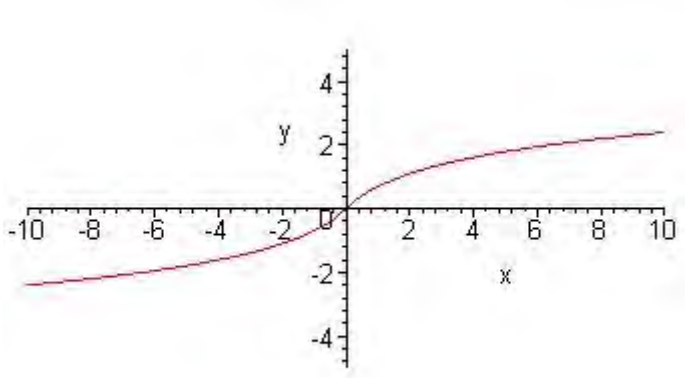
Afin de pouvoir modifier la présentation d'un cours, nous avons choisi de nous appuyer sur deux types d'informations.
Tout d'abord sur la réussite et l'échec aux exercices de l'étudiant en fonction des trois présentations du cours. Ceci correspond aux champs *nbReussiTxt*, *nbReussiGr*, *nbReussiAnim*,

nbEchoueTxt, *nbEchoueGr* et *nbEchoueAnim* de la table *infoetudiant*. Nous dénombrons dans ces champs les exercices réussis et les exercices échoués en fonction des différents types de cours qui ont été présentés à l'étudiant.
Le deuxième critère est le temps passé sur les pages de cours, toujours en fonction de sa présentation. Pour cela, on fait le cumul des secondes passées par l'étudiant sur chacun des trois types de cours (*tempsTxt*, *tempsGr*, *tempsAnim*).

Pour interpréter ces informations, nous avons ensuite dû écrire deux fonctions dont les résultats sont utilisés dans l'algorithme afin de modifier la présentation de l'étudiant.

La première fonction, *calculEvalExo,* nous informe de façon pondérée de la réussite ou de l'échec d'un étudiant aux exercices en fonction du type de cours qui lui a été présenté. Nous utilisons une composée de la fonction ln (logarithme népérien) afin de traiter les valeurs négatives.
Ainsi, nous calculons d'abord la différence entre le nombre d'exercices réussis et échoués pour un type de page. Puis nous renvoyons une valeur pondérée comme énoncé ci-dessous :

- $x \geq 0 \Rightarrow \ln(x+1)$
- $x < 0 \Rightarrow -\ln(1-x)$



On constate ainsi que les valeurs augmentent de moins en moins vite lorsque la différence entre les exercices réussis et échoués augmente.
La deuxième fonction, *calculMoyenneTemps*, indique le temps moyen passé sur les pages de cours du type passé en paramètre :

$$\frac{\dfrac{temps\ total\ d'un\ type}{nb\ réussi\ type + nb\ échoué\ type}}{5}$$

Dans un premier temps, nous pensions ne prendre en compte que le nombre de page d'exercices réussis mais pour affiner les calculs, nous avons décidé de compter le nombre d'exercices réussis et échoués. Ainsi nous faisons une différence entre un étudiant qui répond correctement à tous les exercices qui lui sont proposés et un étudiant qui a juste la moyenne.

Chaque page d'exercices comporte cinq exercices. Quelque soit le résultat obtenu, la somme du nombre d'exercices réussis et échoués d'un type de page donnée est un multiple de cinq. En divisant par cinq, nous obtenons le nombre de pages d'exercices parcourues qui

correspond donc également au nombre de pages de cours. En divisant le temps total par le nombre de pages de cours, nous obtenons le temps moyen par page de cours d'un type donné. Plus simplement, nous pouvons utiliser la formule équivalente suivante :

$$\frac{temps\ total\ d'un\ type * 5}{nb\ réussi\ type + nb\ échoué\ type}$$

Intégration de l'algorithme dans le site

La phase d'intégration de l'algorithme dans le site, réalisée par Patrice Behem et Jérôme Guillot, s'est étendue sur deux jours de travail à plein temps, mais de nombreuses modifications ont été apportées par la suite.

Le développement de l'algorithme a eu lieu en parallèle avec la construction du site. Il a donc été nécessaire de les réunir. Cette intégration a surtout nécessité la compréhension du code de la partie réalisée par l'autre équipe afin de connaître l'environnement dans lequel on devait insérer l'algorithme. Certaines parties du code ont été modifiées pour le rendre compatible avec le site, comme par exemple certaines variables globales de la page (notamment les variables utilisées pour une connexion à la base de donnée) et également la modification de fonctions ou la suppression de celles définies dans les deux parties.

Conclusion sur l'implémentation de l'algorithme

La programmation en php de l'algorithme s'est avérée au départ assez pratique, surtout au niveau de la facilité de son apprentissage. Mais au fur et à mesure, le code s'est étoffé et c'est un véritable programme que l'on a dû coder, et pas seulement une gestion dynamique de l'affichage d'une page Web. Bien que tous les outils requis existaient en php, l'utilisation d'un langage plus fourni et plus contrôlé, comme le Java (avec le jsp), aurait peut-être été plus pratique.

2.7 L'application

Pour mettre en place notre application, nous avons utilisé le logiciel *EasyPHP 1.7* qui regroupe le système de gestion de base de données *MySQL*, le langage *PHP* et le serveur Web *APACHE*. Nous n'avons testé le projet qu'avec un serveur tournant sous Windows, le fonctionnement sous Linux et MacOs étant secondaire.

Les pages de script sont interprétées côté serveur, puis le résultat est transmis au client. L'utilisateur ne peut donc pas accéder au code *PHP*. Cette interprétation côté serveur offre la possibilité de modifier le contenu du site en fonction de l'utilisateur notamment en accédant aux valeurs présentes dans la base de données. Nous détaillerons dans les sous-sections suivantes la structure de la base de données et l'architecture du site Web.

2.7.1 Structure de la base de données

La structure de notre base de données *elearning* est la suivante :

Cours (nomCours, auteur, nbChapitres)
CoursEtudiant (nomCours, login, niveau, idPageCourante)
Etudiant (login, mdp, email, idPresentation, idSession)
Exercice (nomExo, idPage, reponse, difficulte, nbreussi, nbpropose)
Page (idPage, nomPage, nomCours, idPresentation, niveau, tempsprévu)
Lien (idLien, idPageSrc, idPageDest, retour, W, S, F)
HistLienEtudiant (idLien, login, date, temps)
InfoEtudiant (login, nbReussiTxt, nbReussiGr, nbReussiAnim, nbEchoueTxt, nbEchoueGr, nbEchoueAnim, tempsTxt, tempsGr, tempsAnim)
Presentation (idPresentation, type)
HistFitness (idLien, date, fitness)
HistNbEtapes (login, nomCours, date, nbEtapes, fini)

La table qui suit n'a pas de relation directe avec un étudiant ou un cours, mais avec l'algorithme mis en œuvre.

Variable (nomVariable, valeur)

Il s'agit d'une base de données relationnelle; les clés primaires sont soulignées en trait plein, et les clés étrangères le sont en pointillés.
Le détail de la structure de chacune de ces tables est donné dans les sous-sections qui suivent. Nous avons déterminé une structure de base, qui n'a eu de cesse d'évoluer tout au long de notre projet, suivant les besoins de chaque binôme. C'est pourquoi il nous serait difficile d'attribuer à une personne en particulier la réalisation de cette tâche.

Table Cours

Cette table contient tous les cours disponibles sur le site Web : ils sont identifiés par leur nom (*nomCours*), et ont un auteur ainsi qu'un nombre de chapitres. Ces derniers sont numérotés de 0 à *nbChapitres*. Chaque cours contient des pages ayant un niveau croissant selon l'avancée dans le cours.

Table CoursEtudiant

Cette table permet de stocker le *niveau* atteint par un étudiant pour chaque cours. Les cours non encore consultés par l'étudiant n'apparaissent pas dans cette table. Les cours commencés par l'étudiant ont un *niveau* entre 1 et *nbChapitres*, et ceux qu'il a achevés ont un *niveau* égal à *nbChapitres* + 1. Le champ *idPageCourante* permet de connaître la dernière page de ce cours visitée par l'étudiant.

Table Etudiant

Cette table comporte les informations relatives à chaque étudiant inscrit sur notre site. Lors de son inscription, il choisit un login, un mot de passe, et fournit une adresse email. Il choisit également un format de présentation préféré pour les cours : soit uniquement du texte, soit du texte et des graphiques, soit du texte et des animations. S'il n'a pas de préférences, on lui attribue la première possibilité. Cette information est stockée dans la variable *idPresentation*, qui est une clé étrangère vers la table **Presentation** décrite plus loin. Le champ *idSession* correspond à l'identifiant de la session courante pour un étudiant. Il permet de ne pas avoir deux sessions ouvertes en même temps pour un même étudiant.

Table Exercice

Cette table recense tous les exercices, identifiés par leur nom. Chaque exercice est un QCM et appartient à une page (*idPage* comme clé étrangère). Le champ *reponse* est la bonne réponse au QCM. Le champ *difficulte* correspond au niveau de difficulté de l'exercice : nous avons supposé qu'il y avait 5 niveaux de difficultés pour les exercices; ce niveau est calculé en fonction des champs *nbreussi et nbpropose*, qui sont respectivement le nombre de fois où l'exercice a été réussi, et le nombre de fois où l'exercice a été proposé.

Table Page

Cette table contient toutes les pages disponibles sur le site, avec leur nom, le cours auquel elles appartiennent, leur format de présentation, leur niveau dans le cours, ainsi que le temps moyen nécessaire à sa compréhension (prévu par l'enseignant).

Table Lien

Cette table contient les liens entre les pages de cours disponibles sur notre site. Un lien relie une page source *idPageSrc* à une page destination *idPageDest*, et comporte des paramètres spécifiques à chaque lien permettant à l'algorithme de fonctionner (*W*, *S* et *F*). Le champ *retour* vaut 1 s'il s'agit d'un lien de retour, et 0 sinon.

Table HistLienEtudiant

Cette table permet de conserver l'historique du parcours de chaque étudiant dans les cours. Nous avons choisi de créer l'historique à partir des liens entre les pages (*idLien*) et non pas à partir des pages visitées, ce qui permet de connaître immédiatement les étapes suivies pour arriver à un certain niveau. On garde comme clé primaire la date à laquelle l'étudiant a parcouru un lien pour pouvoir conserver tous les passages d'un étudiant sur ce lien. On conserve également le temps passé par l'étudiant sur la page destination du lien.

Table InfoEtudiant

Cette table contient des informations relatives à l'étudiant, et permet de conserver au fur et à mesure de son cheminement, le nombre d'exercices qu'il a réussi et qu'il a échoué en fonction de la présentation du cours qui précédait. Ceci est conservé dans les champs *nbReussiTxt*, *nbReussiGr*, *nbReussiAnim*, *nbEchoueTxt*, *nbEchoueGr*, *nbEchoueAnim*. Dans les champs *tempsTxt*, *tempsGr* et *tempsAnim*, on fait le cumul des secondes passées par l'étudiant sur chacun des trois types de cours.

Table Presentation

Cette table associe un numéro de format de présentation à un nom. Elle contient les 4 enregistrements suivants :

1 -> txt (uniquement du texte)
2 -> ill (du texte et des graphiques)
3 -> anim (du texte et des animations)
0 -> exo (c'est une page d'exercice, elle n'a pas de format de présentation spécial)

Table Histfitness

Cette table contient l'historique des fitness de chaque lien enregistré dans la base. Elle est utile pour la phase de test et d'analyse, afin de pouvoir récupérer l'évolution de la fitness d'un lien souhaité.

Cette table contient les champs suivants : *idLien* l'identificateur du lien, *date* pour pouvoir ordonner les fitness, et *fitness* la valeur de la fitness du lien.

Table HistNbEtapes

Cette table contient l'historique du nombre d'étapes effectuées par les étudiants pour terminer un cours. Tout comme la table **HistFitness**, elle est utile pour la phase de test et d'analyse ; nous pouvons ainsi récupérer l'évolution de ce nombre d'étapes.
Cette table contient *login* le login de l'étudiant, *nomCours* le nom du cours, *date* pour pouvoir ordonner les nombres d'étapes, *nbetapes* le nombre d'étapes effectuées par l'étudiant donné pour terminer le cours donné, et *fini* un booléen permettant de savoir si le cours terminé.

Table Variable

Cette table permet d'associer un nom de variable à une valeur. Ces variables sont utilisées dans l'application de l'algorithme.

Voici le détail de ces variables :

*Omega1***,** *Omega2* et *Omega3* sont les pondérateurs des coefficients W, S et F de la fitness.

Epsilon est la plus petite valeur que peut prendre une fitness.

*OmegaPreferencePres***,** *OmegaTempsMoy* et *OmegaEvalExo* sont les pondérateurs des évaluateurs.

Alpha1 et *Alpha2* sont les valeurs d'incrémentation des valeurs de S ou F en cas de succès ou d'échec (*Alpha1* pour le succès et *Alpha2* pour l'échec).

Nblienmodif représente le nombre de liens modifiés lorsqu'un étudiant réussit ou échoue un exercice. Il s'agit des modifications des S et des F des liens précédant l'exercice.

SiPresEq est la valeur prise par l'évaluateur PreferencePresentation en cas de correspondance entre la présentation d'un lien et celle contenue dans le profil de l'étudiant.

SiPresDiff est la valeur prise par l'évaluateur PreferencePresentation en cas de différence entre la présentation d'un lien et celle contenue dans le profil de l'étudiant.

ChoixAlgo est le choix de l'algorithme de sélection utilisé :

1 pour la procédure de sélection par roulette
2 pour la procédure de sélection par rang automatique
3 pour la procédure de sélection par rang manuel
4 pour la procédure de sélection par tournoi

*Rangmanuelfirst***,** *Rangmanuelsecond* et *Rangmanuelother* sont des valeurs utilisées dans la procédure de sélection par rang manuel.

Tournoidiv représente le nombre d'équipes sélectionnées dans la procédure de sélection par tournoi.

CoeffChangePref est la valeur du coefficient k dans la formule du changement de profil.

NbPagesChangePref représente le nombre de pages que l'étudiant doit parcourir avant que son profil soit susceptible d'être modifié.

ChangePreference est un drapeau qui permet de déterminer si on accepte les changements de profils automatiques : cette variable vaut 1 si on les accepte, et 0 sinon.

2.7.2 Architecture du site Web

Cette partie a été réalisée en binôme par Corinne Albarelli et Laetitia Joly pendant 7 jours à plein temps.

Contenu et mise en forme

Les pages du site Web sont codées dans les langages *HTML*, *PHP* et *JavaScript*, *PHP* étant utilisé principalement pour les connexions à la base de données et la différenciation des utilisateurs, et *JavaScript* pour le rendu dynamique.

Une page du site Web est en fait l'inclusion de plusieurs pages *PHP*, puisque certains éléments sont communs à toutes les pages. La disposition de ces éléments se fait à l'aide de tableaux. On a également créé une feuille de style *CSS* pour avoir un affichage plus convivial.

Fonctionnalités

Menu adapté au visiteur

Le site présente un menu différent selon l'état de connexion de l'étudiant qui parcourt le site. S'il est connecté, il pourra accéder à des données le concernant, sinon les données seront anonymes.

Inscription d'un étudiant

L'étudiant est amené à remplir un formulaire. Les informations fournies sont stockées dans la base de données pour une utilisation future. Ces informations correspondent aux identifiants de connexion (login et mot de passe), à l'adresse email et à la préférence de présentation. Une fois l'inscription réalisée, l'étudiant est considéré comme « membre » du site Web.

Connexion / déconnexion d'un membre

Pour se connecter, le membre entre ses identifiants de connexion aux endroits voulus. Pour se déconnecter, il doit simplement appuyer sur le lien de déconnexion.

Perte des identifiants

Si un membre ne se souvient plus de ses identifiants de connexion, il peut demander à les afficher sur le site (voir la section *« Choix d'implémentation »*) en entrant son adresse email.

Affichage et changement des préférences d'un membre

Pour avoir un récapitulatif de ses préférences, le membre doit se rendre sur la page « Mes préférences ». Si une information ne lui convient plus, il a la possibilité de la modifier en remplissant un formulaire : il choisit tout d'abord les préférences qu'il désire modifier en cochant les cases ad hoc. Seuls les champs correspondants à ces choix lui seront proposés afin de les modifier.

Affichage de la liste des cours

Un étudiant a la possibilité, avant se d'inscrire ou de se connecter, de consulter la liste des cours disponibles sur le site Web. Ceci lui permet de savoir si les cours proposés lui conviennent.

Un membre peut accéder à un récapitulatif personnel des cours étudiés en allant sur la page « Mes cours ». Elle présente la liste des cours actuellement étudiés avec pour chacun un lien pour continuer l'apprentissage, la liste des cours achevés et la liste des cours qu'il n'a jamais étudiés.

Aide

Un étudiant peut consulter l'aide. Cette aide diffère selon que l'étudiant est connecté ou non puisque le contenu du site change dans ce cas là.

Gestion des erreurs

En cas d'erreurs de l'utilisateur au cours des différentes étapes décrites précédemment (remplissage incorrect d'un champ ou omission), les erreurs sont identifiées et signalées précisément. L'utilisateur est alors invité à recommencer la procédure.

Page d'administration

Seul l'administrateur (de login « *admin* ») peut y accéder. Elle permet de réaliser certaines opérations sur la base comme l'initialisation ou la suppression des étudiants, l'initialisation des paramètres de l'algorithme.

Choix d'implémentation

Accès au serveur

Le site Web peut être visité sur un ordinateur autre que l'ordinateur serveur puisque nous n'avons pas utilisé le mot « localhost » dans les formulaires, mais une fonction *PHP* qui permet de récupérer l'adresse IP du serveur Web.

Connexion à la base de données

En ce qui concerne les connexions à la base de données, nous avons décidé d'en créer une à chaque fois que cela était nécessaire. Et pour plus de sécurité, nous avons créé un utilisateur « admin » avec les privilèges « select », « insert », « update », « delete » et « file ».

Les sessions

Nous avons mis en place le mécanisme des sessions, c'est-à-dire qu'une session est ouverte à chaque connexion d'un membre. Ceci permet de définir des variables qui prendront des valeurs différentes en fonction de l'utilisateur connecté. Ces variables sont utilisables pendant toute la durée de la session, entre la connexion et la déconnexion, et sont spécifiques à l'étudiant connecté, un étudiant ne peut pas accéder aux variables d'un autre étudiant.

Envoi d'un courriel

Nous avions pensé envoyer un courriel à l'étudiant pour confirmer son inscription, la modification de ses préférences ou pour lui rappeler ses identifiants de connexion, mais nous n'avons pas pu mettre en place ce mécanisme étant donné que le serveur tourne en local. Nous avons donc mis ce code en commentaire au cas où on voudrait s'en servir ultérieurement. Pour le rappel des identifiants, nous avons décidé de les afficher sur une page même si cette solution pose des problèmes de sécurité.

Gestion de la connexion

Pour éviter qu'un membre n'ait deux sessions ouvertes en même temps, nous avons décidé d'ajouter le champ *idSession* dans la table **etudiant**. Ce champ prend la valeur de la session courante (renvoyé par *session_id()*) lorsque celle-ci est ouverte (l'étudiant se connecte). Ainsi, lors de l'affichage d'une page, nous pouvons tester la valeur de ce champ : si elle est différente de l'id de session courante, alors la connexion est interrompue et une erreur est générée. Ceci permet bien d'éviter l'ouverture de deux sessions, puisque l'ouverture de la deuxième session entraînera automatiquement la fermeture de la première.

Gestion de l'usurpation d'identité

Etant donné que le login est visible dans l'URL, un utilisateur mal intentionné pourrait usurper l'identité d'un étudiant. Pour éviter cela, le mot de passe est conservé dans une variable de session lors de la connexion ou de l'inscription, et pour chaque page affichée, nous testons si le mot de passe correspondant au login est le même que celui de la variable de session. Si ce n'est pas le cas, alors la connexion est interrompue et une erreur est générée.

Utilisation de la technologie JavaScript

Le langage JavaScript est utilisé plusieurs fois dans le site Web pour avoir un affichage dynamique. Cependant il faut que le navigateur accepte ce type de technologie, ce qui n'est pas toujours le cas.

Difficultés rencontrées

Nous avons eu un problème au niveau des variables. En effet, nous voulions créer des variables globales communes à tous les étudiants et disponibles sur toutes les pages du site. Or en *PHP*, les variables sont utilisables uniquement dans le script dans lequel elles ont été définies, donc la notion de variable globale n'existe pas. Pour remédier à ce problème, nous avons créé dans la base de données une table regroupant toutes les variables associées à leur valeur.

2.8 Les tests

Pour fournir des données à l'algorithme et régler au mieux ses paramètres nous avons dû tester le site. Mais ne pouvant pas demander à un nombre important de personnes de s'inscrire, nous avons décidé de créer des robots qui simulent le comportement d'étudiants. Nous avons également fait appel à quelques personnes pour tester l'interface.

2.8.1 Les testeurs

Nous avons demandé à quelques personnes extérieures à la Maîtrise Informatique de tester le site Web. Grâce à leurs remarques, nous avons pu l'améliorer et corriger quelques erreurs de fonctionnement. L'aide de testeurs non informaticiens nous a également été utile pour voir si l'utilisation du site Web était triviale ou si certaines fonctionnalités étaient mal comprises.

2.8.2 Les « robots étudiants »

Pour tester le bon fonctionnement de l'algorithme, nous avons mis en place des robots, lancés à partir d'un programme java indépendant du site Web. Un robot représentant un étudiant, il devra s'inscrire sur le site, parcourir le cours le nombre de fois voulu, et se déconnecter. Nous détaillerons dans la section suivante les fonctionnalités.
Cette partie a été réalisée en binôme par Corinne Albarelli et Laetitia Joly pendant 5 jours à plein temps.

Fonctionnalités

Initialisation et lancement des robots

Un robot possède un identifiant correspondant à la fois au login et au mot de passe (sous la forme robot1, robot2, …, robotn), une présentation de départ, et une présentation préférée vers laquelle il devra être redirigé. Ces présentations sont déterminées de façon aléatoire lors de la création du robot. Le testeur détermine le nombre de robots qu'il souhaite lancer ainsi que le cours utilisé pour le test. Les robots sont ensuite lancés en parallèle.

Simulation du parcours d'un étudiant

Le robot s'inscrit sur le site Web avec ses identifiants de connexion. Il débute l'apprentissage du cours voulu, et passe aux pages suivantes en fonction des données envoyées par l'algorithme. Le robot passera moins de temps sur une page du format de sa

présentation préférée, et réussira mieux les exercices lorsqu'il vient d'une page de sa présentation préférée. Ces deux paramètres permettent d'avoir des robots avec des comportements assez variés. Nous avons également choisi 4 pages du cours pour lesquelles le robot échoue dans tous les cas, pour voir si les paramètres collectifs sont pris en compte.
Le robot peut apprendre un cours plusieurs fois de suite. Après avoir terminé, il se déconnecte.

Traitement des résultats

Lorsque tous les robots ont fini leur exécution, un fichier texte est généré afin de conserver la présentation de départ et la présentation préférée de chaque robot. Ce fichier est utilisé par une autre application afin de traiter les résultats : en effet, il est ainsi possible de vérifier que le robot a bien été redirigé vers les pages du format de sa présentation préférée si celle-ci est différente de le présentation de départ. De plus, il est possible de lancer plusieurs fois le même robot sur un même cours, afin de vérifier qu'il obtient de meilleurs résultats s'il a déjà appris le cours.

Choix d'implémentation

Utilisation de threads

Chaque robot correspond à une thread. Ces dernières sont lancées en parallèle par une thread principale permettant leur synchronisation. Lorsqu'une thread robot a fini son exécution, elle notifie la thread principale. Le traitement des résultats s'effectue après la notification de la thread principale par toutes les threads robot. Il est possible de lancer plusieurs fois les mêmes robots sur un même cours : dans ce cas, on utilise toujours une thread par robot. Chaque thread notifie la thread principale à chaque fois qu'elle a terminé un passage sur le cours. Lorsque toutes les threads ont fini un passage sur le cours, la thread principale attend qu'elles aient terminé le passage suivant, et ainsi de suite jusqu'à la fin.

Appel de l'algorithme

Pour débuter un cours, le robot appelle l'algorithme avec le nom de ce cours. A partir de là, le robot appelle l'algorithme avec le numéro de la page courante pour connaître la page suivante.
A chaque appel de l'algorithme, ce dernier génère un fichier texte portant le nom du robot, et contenant les informations sur la prochaine page à visiter : l'identifiant de cette page, son format de présentation et le temps prévu pour une page de cours, et, pour une page d'exercices, on y ajoute le nombre d'exercices et leur nom. Le robot lit ce fichier, et appelle à nouveau l'algorithme avec la nouvelle page. Ceci est effectué en boucle jusqu'à ce que le robot atteigne la dernière page du cours : l'algorithme lui renvoie alors -1 comme page suivante.
Le format de présentation ainsi que le temps prévu servent au paramétrage des robots.

2.8.3 L'application de test

L'application de test a été réalisée en grand partie par Patrice Behem, aidé de Jérôme Guillot pendant une durée effective de 5 jours.

Afin de pouvoir récupérer et visualiser les résultats des tests faits par les robots étudiants, nous avons choisi de programmer une application qui, à l'aide de requêtes dans la base de données de notre programme, affiche des graphiques. L'intérêt d'une telle application

étant de rendre visibles les évolutions suivies par l'algorithme ACO. Les graphiques affichés nous ont également aidé lors de la phase d'analyse et de paramétrage de l'algorithme. Il a donc été important de choisir des graphiques pertinents apportant de réelles informations desquelles nous avons pu déduire les améliorations de paramétrage à apporter à l'algorithme.

Cette application nous donne trois types d'informations nécessaires à notre phase d'analyse :

1. **L'évolution de la valeur d'une fitness d'un lien**

Il est effectivement possible dans cette application de donner en paramètre un ou plusieurs liens dont nous souhaitons voir l'évolution de la fitness en fonction du nombre de passages sur ce lien. A l'aide d'une requête dans la table d'historique des fitness des liens, l'application affiche les graphiques de l'évolution des fitness des liens donnés. Ce faisant, il nous est alors possible de comparer l'évolution des liens d'un même niveau pour savoir si au fil du temps le lien ayant le plus de chances d'être proposé aux étudiant change. Il nous est également possible de visualiser la réactivité de l'algorithme en mesurant le nombre de passages qu'il faut avant de voir un changement. A noter toutefois que pour ce graphique nous ne prenons en compte que les paramètres collectifs de la fitness, afin que les paramètres personnels de chaque étudiant (qui sont tous différents) ne viennent pas « polluer » les informations que nous devons analyser ici.

2. **Le nombre d'étapes pour finir un cours**

Il est également possible dans cette application de donner en paramètre un étudiant et un cours pour voir l'évolution du nombre d'étapes mises par cet étudiant pour terminer ce cours. Cela peut paraître abstrait, mais le fait de lancer plusieurs fois le même robot étudiant sur le même cours en récupérant à chaque fois le nombre d'étapes effectuées pour terminer ce cours, permet de vérifier que l'algorithme ACO s'adapte bel et bien à cet étudiant. En effet, au fur et à mesure de ses différents passages sur ce même cours, les calculs des fitness des liens en fonctions des paramètres collectifs, mais surtout ici individuels, vont s'affiner. Ainsi lorsque l'algorithme s'adapte à l'étudiant conformément à nos attentes, nous devrions observer une diminution du nombre d'étapes pour terminer ce cours.

3. **L'adaptation de l'algorithme aux étudiants**

Enfin, cette application permet de récupérer les informations nécessaires sur tous les robots étudiants pour vérifier qu'à la fin de la phase de test, la préférence de présentation des robots étudiants est bien conforme à la préférence réelle choisie pour chaque robot étudiant. Autrement dit, nous vérifions que les éventuelles modifications de profil d'étudiant qui sont nécessaires ont bien eu lieu. Ce résultat est alors affiché sous forme d'un pourcentage.

2.8.4 Tests et analyse des résultats

Dans un premier temps, nous avons réalisé une phase de tests en ne prenant en compte que les paramètres personnels pour pouvoir, d'une part, tester les robots et, d'autres part, avoir de premiers résultats concernant le fonctionnement de l'algorithme. Et dans un

deuxième temps, nous avons modifié les robots pour prendre en compte les paramètres collectifs.

Tests avec seulement les paramètres personnels

Nous avons effectué deux types de tests. Le premier consiste à faire parcourir le cours à un grand nombre de robots (660) mais un nombre réduit de fois (3). A l'inverse le second type de test prend en compte peu de robots (7), mais leur fait parcourir de nombreuses fois (environ 60) le cours. Nous n'avons pas pu combiner les deux types de tests car cela prenait trop de temps. En effet chacun des tests a nécessité plus d'une nuit.

- **Analyses des préférences**

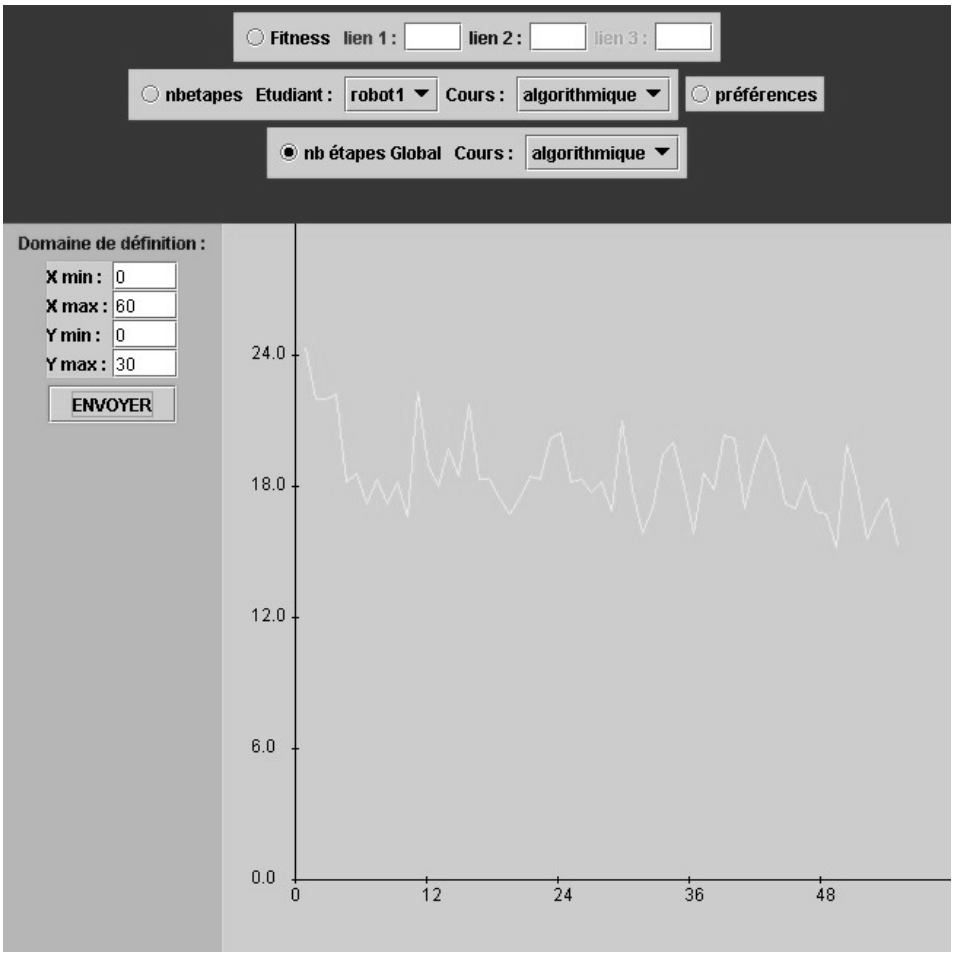
Dans un premier temps, nous avons observé les modifications des préférences des robots. Afin de vérifier l'adéquation de nos changements, nous avons dû analyser les statistiques obtenues :

  - Pour 68.12% des robots la préférence a été convenablement modifiée dans la base de données. C'est-à-dire que celle de départ n'est plus la même que celle à la fin du parcours, et que cette dernière correspond à la préférence pour laquelle le robot réussit le mieux.
  - Pour 0.15% des robots la préférence n'a pas été changée comme elle aurait dû. L'algorithme n'a pas attribué à ces robots la préférence avec laquelle ils réussissent de façon optimale.
  - Pour 0.60% des robots la préférence n'a pas été modifiée alors qu'elle ne correspond pas à celle avec laquelle ils réussissent le mieux.
  - Pour 30.68% des robots la préférence n'a pas été changée et correspond bien à celle avec laquelle ils réussissent le mieux (elle n'avait donc pas à être changée).
  - Pour 0.45% des robots la préférence ne devait pas être modifiée mais a quand même subi un changement.

Nous constatons que l'algorithme attribue la plupart du temps la bonne préférence aux robots : en effet, seulement 1,2% des robots lancés a été mal redirigé (soit 6 sur 600 ici).

- **Evolution du nombre d'étapes**

Dans un deuxième temps, nous avons étudié le nombre d'étapes qu'il a fallu à un étudiant pour parcourir la totalité d'un cours. Il s'agit, dans nos tests, du nombre de pages vues par un robot. Nous avons ainsi pu constater son évolution lorsqu'il parcourt plusieurs fois le même cours. Chaque passage d'un robot simule un nouvel étudiant qui possède les mêmes préférences que celui d'origine, et réussit avec les mêmes probabilités.
C'est le second type de test qui nous a permis d'établir ces résultats. Pour illustrer ces derniers, nous avons tracé la courbe qui montre le nombre moyen d'étapes des sept robots pour chaque parcours du cours.



Nous constatons que la courbe est globalement décroissante ce qui indique un nombre d'étapes moyen en diminution.

Tests avec les paramètres collectifs

A l'aide de l'application de test, nous avons lancé une série de tests permettant d'avoir une première impression du comportement de l'algorithme. Nous nous sommes rapidement rendu compte de l'importance et de la difficulté du paramétrage.

Premiers tests, et premières difficultés

Pour les premiers tests, les valeurs données aux différents paramètres de l'algorithme ACO étaient plus intuitives que réellement réfléchies. Les premiers résultats étaient donc peu satisfaisants, et nous avons commencé la phase de paramétrage.

Nous nous sommes tout de suite aperçu des nombreuses difficultés liées à ce paramétrage :

- Chaque changement, même minime, d'un paramètre entraîne des modifications importantes du comportement de l'algorithme.
- Il faut garder une cohérence globale entre les différents paramètres dont le nombre est important.
- Le bon fonctionnement de l'algorithme dépend principalement de sa phase d'apprentissage et donc de l'utilisation de celui-ci par un grand nombre d'étudiants. Nos tests avec des robots étudiants ne nous permettent pas de lancer de nombreux

parcours simultanés. Chaque test nécessiterait une centaine de passage pour être vraiment pertinent, ce qui prend alors un temps considérable (plusieurs heures). Nous ne pouvons donc pas relancer ces centaines de parcours à chaque changement de paramètre.

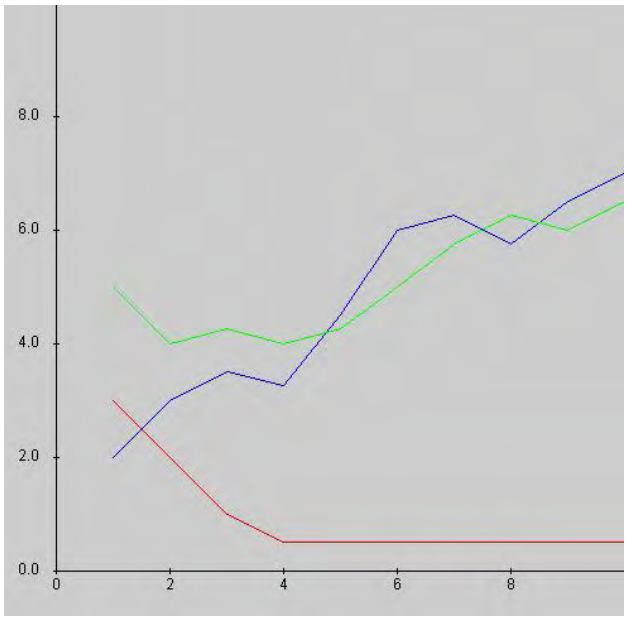
- L'analyse des résultats obtenus après chaque test est également assez fastidieuse. Pour vérifier si le comportement de l'algorithme correspond à nos attentes, nous devons analyser les graphes de l'application de tests, les traces générées dans des fichiers log, et la base de données.
- Notre temps de travail ayant débordé par rapport à notre planning initial, nous n'avons pas eu assez de temps pour effectuer tous les tests nécessaires.

Paramétrage de l'algorithme ACO

Les différentes variables paramétrables dans l'algorithme sont : les pondérateurs dans la formule de la fitness, la valeur minimale d'une fitness, les valeurs retournées par certains évaluateurs. Sans oublier le choix de la procédure de sélection.

Pour tester, nous avons choisi la procédure de sélection par rang manuel, car elle permet de mieux contrôler d'éventuelles croissances excessives des fitness.

Afin de vérifier que notre paramétrage est correct, nous devons alors observer que le parcours des étudiants est bien conforme à celui souhaité. Nous avons donc créé des pages sur lesquelles tous les robots étudiants échouaient toujours. Nous devons observer au fur et à mesure des passages des robots, que ces derniers (même ceux dont la préférence correspond à la présentation des pages interdites) vont bien passer par un autre chemin. Dans ce cas là, la partie collective de la fitness des liens doit prendre le pas sur la partie personnelle. On doit également remarquer que la partie collective de la fitness de la page « interdite » doit tomber assez rapidement à son minimum.



On voit ici l'évolution de la partie collective des fitness de trois liens concurrents. Le graphe en rouge représente le lien menant à la page « interdite ». Sa fitness diminue bien.

Au départ, l'évolution des fitness des liens était trop rapide et pouvait atteindre des valeurs trop élevées. Le but de ces paramétrages était donc de trouver un juste milieu pour

obtenir un algorithme dont la réactivité ne soit pas trop élevée, sans pour autant qu'elle devienne trop faible.

De plus, un juste milieu devait être trouvé concernant le poids de la partie personnelle de la fitness des liens, par rapport à celui de la partie collective.

Pour pondérer la partie personnelle, nous devons jouer avec les variables *OmegaPreferencePres*, *OmegaTempsMoy* et *OmegaEvalExo*. En faisant en sorte que la somme des paramètres personnels n'atteigne pas de valeurs extrêmes ; pour donner un ordre de grandeur, cette somme pourrait osciller entre 1 et 4.

De la même manière, pour pondérer la partie collective, nous devons jouer avec les variables *Omega1*, *Omega2* et *Omega3*, à la différence que la somme des paramètres collectifs n'est pas bornée. En effet, elle est appelée à être augmentée ou diminuée à chaque réussite ou échec. De plus, nous devons prendre en compte les variables *Alpha1* et *Alpha2*, qui sont les valeurs d'incrémentation des coefficients S et F, de manière à rendre très progressive ces incrémentations. Le but étant de ne pas prendre trop vite le pas sur la partie personnelle.

A noter que nous avons fixé comme critère principal de bon fonctionnement de l'algorithme, l'évolution du nombre d'étapes effectuées par les étudiants pour terminer un cours. Ainsi, une diminution de ce nombre d'étapes indique que l'algorithme s'adapte bien à l'étudiant en le faisant réussir de mieux en mieux.

2.8.5 Conclusion

La phase de tests et d'analyse n'a pas pu être menée jusqu'à son terme par manque de temps. On peut toutefois conclure que chaque paramétrage différent a bien fait évoluer le comportement de l'algorithme comme souhaité. Cependant nous n'avons pas pu effectuer de paramétrage à grande échelle permettant d'obtenir une cohérence globale de tous les paramètres. De plus le comportement des robots étudiants n'est pas véritablement humain, et il ne permet pas de tester les paramètres collectifs correctement. Il aurait fallu pour cela intégrer le fait qu'indépendamment de la présentation, certains cours sont plus faciles à étudier et qu'ils entraînent donc un meilleur taux de réussite. Le moyen utilisé pour contourner ce problème a été de créer le concept des pages « interdites ».

2.9 Les extensions possibles

N'ayant eu que quelques mois pour réaliser le projet, nous sommes allés à l'essentiel en mettant en place les bases d'un projet conséquent. Cependant nous avons réfléchi à ce qui pourrait être nécessaire, voire indispensable dans une vraie application de e-learning.

Création d'un éditeur de cours

Dans l'état final de notre projet, si nous voulons ajouter, supprimer ou modifier un cours, nous sommes obligés d'avoir accès aux dossiers du site Web et d'avoir les droits de modification de la base de données, ce qui évidemment n'est pas sécurisé dans le cas où notre projet serait réellement utilisé. Pour éviter cela, il serait intéressant de créer un éditeur de cours qui permettrait à des professeurs de proposer et de gérer leurs cours sans qu'ils aient besoin d'avoir des connaissances en base de données ou d'avoir certains privilèges. Cet

éditeur pourrait être externe à l'application avec un bouton permettant de le lancer depuis le site Web ou bien intégré directement dans celui-ci. Il possèderait au minimum les deux fonctionnalités suivantes.

Téléchargement des pages du cours

Les pages des cours étant dans le langage HTML, il faudrait prévoir de ne garder que le contenu de leur corps (entre les balises <*BODY*> et <*/BODY*>) et de changer l'extension en *php* pour qu'elles puissent être incluses dans le site Web. Ceci n'est qu'une vague idée du travail à réaliser puisque nous n'avons pas approfondi la réflexion sur ce point.

Description du schéma du cours

Nous avons pensé à deux manières de décrire le schéma du cours, la deuxième offrant plus de possibilité que la première. Ceci est valable aussi bien pour l'ajout d'un nouveau cours que pour la modification d'un cours déjà présent et pas forcément ajouté par le même professeur.

Sans visualisation graphique :

Un professeur ayant peu de connaissances en informatique serait amené à remplir une sorte de formulaire dans lequel il indiquerait les liens entre les chapitres et les exercices, le niveau des chapitres et toutes les informations nécessaires à la construction du cours.

Un professeur confirmé décrirait le schéma dans un fichier XML dont la syntaxe serait proposée par l'éditeur.

Avec visualisation graphique :

Le schéma du cours serait représenté sous forme de graphe, le professeur aurait la possibilité de tracer les liens entre les chapitres et les exercices et d'éditer une feuille de propriétés des liens et des pages, en double-cliquant sur ceux-ci, pour renseigner le niveau par exemple. Cette méthode a la propriété de pouvoir être utilisée par tous les professeurs quelque soit leurs compétences informatiques.

Amélioration du contenu et de l'aspect du site Web

Diverses améliorations

- ajouter de vrais cours pour pouvoir éventuellement tester le site Web sur un panel d'étudiants,
- faire appel à un Web designer pour l'interface graphique,
- conserver les champs saisis corrects lorsqu'une erreur est signalée pour ne pas avoir à tout recommencer après chaque erreur,
- ajouter la possibilité de supprimer son compte,
- ajouter la possibilité de revoir n'importe quel chapitre d'un cours achevé, sans que l'algorithme en tienne compte.

Page de gestion pour les professeurs

Il faudrait que l'on différencie dans le site un professeur d'un étudiant puisqu'un professeur doit gérer ses cours et ses étudiants, alors qu'un étudiant ne fait qu'apprendre des cours. Pour cela, il faudrait ajouter un champ lors de l'inscription pour savoir dans quelle catégorie se classe l'utilisateur. La page de gestion pour les professeurs présenterait les fonctionnalités listées ci-après:

- l'éditeur de cours vu précédemment,

- la possibilité de visualiser les étudiants qui apprennent ou ont appris ses cours,
- différents modes de recherche d'étudiants, de cours ou de professeurs,
- un résumé de ses cours avec pour chacun la moyenne des étudiants, les chapitres et les exercices qui posent des problèmes, …
- faire passer des étudiants au niveau d'études supérieur ou proposer des cours de soutient à ceux qui ont des difficultés.

Différencier les étudiants selon le niveau d'études

Pour que l'étudiant puisse suivre un parcours d'études plus proche de la réalité, les étudiants devraient être regroupés par niveau d'études, chaque niveau ayant ses propres cours. Nous aurions la possibilité de définir des critères permettant de passer au niveau supérieur, un examen par exemple.

Modifier l'affichage de la liste des cours

On pourrait proposer un affichage de la liste des cours plus évoluer avec, par exemple, la liste des cours du niveau d'études supérieur, mais sans la possibilité de commencer l'apprentissage. On pourrait également établir un mécanisme de recherche pour étudier les cours d'un certain professeur ou les cours d'un niveau donné.

Améliorer l'aide

L'aide présente sur le site ne concerne que son utilisation, mais il serait indispensable dans le cas d'une application réelle d'améliorer cette aide. Nous pourrions, par exemple, créer un forum de discussion qui permettrait un échange entre élèves et professeurs, mettre en place un système d'aide par courriel, chat ou vidéoconférence ou encore réaliser un descriptif détaillé de chaque cours.

Amélioration de l'algorithme

Pour faire évoluer l'algorithme, on peut définir de nouveaux critères personnels. Ces critères peuvent être aussi variés que le choix d'un auteur préféré (si on imagine plusieurs professeurs participants à l'écriture d'un même cours), le choix d'un niveau de difficulté, les choix entre une formation pratique ou théorique, un choix entre différents modes d'apprentissage, etc.…Les possibilités sont nombreuses.

On peut aussi développer l'apprentissage du comportement de l'étudiant en fonction de sa navigation. Pour l'instant nous ne tenons compte que du temps passé sur la page et du résultat qu'il obtient aux exercices qui lui sont proposés.
Mais on peut imaginer, par exemple, que le cours serait composé de liens hypertextes qui permettraient de remonter dans des pages visitées. Si l'étudiant clique sur un de ces liens, on pourrait conclure qu'il n'avait pas parfaitement compris cette page puisqu'il doit la relire.
Ou encore on pourrait prendre en compte la rapidité avec laquelle il répond aux questions.

On peut penser que plus les critères sont nombreux, et plus les cours proposés seront adéquats. En effet, il est probable que les étudiants ne possèdent pas tous les mêmes critères d'apprentissage, cependant on peut espérer que chacun répond à une majorité d'entre eux.

Amélioration du code d'intégration des profils personnels

L'intégration d'un nouveau paramètre dans le profil personnel et d'évaluateurs est expliquée dans le manuel de maintenance. On peut voir que même s'il n'y a pas grand-chose à modifier, nous ne somme pas dans une configuration où une telle intégration est automatique. Il s'agirait d'une amélioration très importante pour permettre une évolution rapide de notre projet.

Nous avons réfléchi à la manière de réaliser ceci. Nous ne l'avons pas fait car ce système s'est avéré en fait difficile et long à mettre en place.

Tout d'abord il faudrait créer une interface qui permette de rajouter un paramètre au profil et un évaluateur de manière automatique. Cette interface pourrait être contenue dans l'éditeur de cours cité précédemment.

L'implémentation d'un nouveau paramètre du profil personnel entraîne la modification de la base de donnée et de quelques fichiers php. Un paramètre du profil personnel est une caractéristique qui doit être définie à la fois chez l'étudiant et dans les pages de cours. Il faut donc modifier la base de donnée pour rajouter ce paramètre du profil dans les tables **etudiant** et **cours**. Modifier la base de donnée de manière automatique n'est pas difficile. Mais il faudrait probablement prévoir une table où serait stocké l'ensemble de ces paramètres de profil pour pouvoir les rajouter à la demande. La difficulté réside plutôt dans l'implémentation du code. Pour que cela marche il faut que les paramètres du profil personnel aient exactement le même comportement dans leurs utilisations. On peut se demander si cela ne risque pas de limiter le profil que l'on peut avoir.
Un autre problème réside dans le changement automatique de la préférence de l'étudiant pour un profil. On change celle-ci à partir de la partie personnelle de la fitness des liens. Mais comme nous n'avons qu'un seul paramètre dans le profil, nous utilisons toute la partie personnelle du calcul de la fitness. Si nous ne voulions utiliser que celle qui correspond à un paramètre du profil en particulier, il faudrait différencier les paramètres du profil dans le calcul de la partie personnelle de la fitness. Or les calculs sont donnés par les évaluateurs. On pourrait croire que chaque évaluateur ne correspond qu'à un paramètre du profil. Mais si nous prenons par exemple le cas de l'évaluateur qui calcule le temps passé sur une page, on s'aperçoit qu'il peut ne pas s'appliquer uniquement à la présentation mais également à tout paramètre qui caractériserait cette page. Il faudrait alors définir une structure qui permette de savoir à quel(s) type(s) de paramètre du profil(s) correspond(ent) chaque évaluateur.
Il faut donc aussi décrire un protocole que respecterait chaque évaluateur. Ce protocole permettrait une insertion rapide des évaluateurs une fois implémentés. On peut tout simplement penser représenter les évaluateurs sous forme d'objets hérités d'une même classe, avec une fonction qui permet d'accéder au résultat renvoyé. Ceci permettrait de rajouter (ou d'enlever) des évaluateurs à loisir dans le calcul de la fitness, puisque tous les résultats qu'ils renvoient sont traités de la même manière. On peut imaginer une méthode qui permettrait de ne pas avoir à toucher au code du programme pour rajouter des évaluateurs. Il suffirait de pouvoir donner l'adresse de l'évaluateur à l'interface qui l'intégrerait dans le code.

Mais l'ajout des évaluateurs ne peut pas être complètement automatique dans le code. Leurs implémentations peuvent ne pas être indépendantes de celles du programme principal. Certains évaluateurs servent à analyser le comportement de l'étudiant. Ils ont donc besoin d'informations sur la navigation de celui-ci. Mais ces informations sont collectées dans le

programme principal, donc on ne peut en définir de nouvelle (qui servirait à de nouveaux évaluateurs) sans modifier du code.

3 Analyse de la réalisation

3.1 Planning

La première partie du planning, du début du TER jusqu'à la pré-soutenance, est reprise dans le tableau ci-dessous ; elle reste inchangée par rapport au cahier des charges : nous avons tous commencé par réfléchir sur le sujet, lire les articles donnés par les encadrants, ... Patrice Behem et Jérôme Guillot ont conçu un modèle de cours d'algorithmique. Pendant ce temps, les quatre autres élèves ont modélisé la base de données et ont réfléchi sur le site Web. Corinne Albarelli a implémenté un évaluateur pour connaître le temps passé sur une page. Pendant que Patrice Behem et Jérôme Guillot ont débuté la programmation de l'algorithme, les autres élèves ont rédigé le cahier des charges. Nous avons ensuite préparé la pré-soutenance tous les six.

Tâche	***Durée***	***Date de début***	***Date de fin***	***Nombre de personnes***
Analyse et spécification (lecture des articles, réflexion sur la modélisation d'un cours, …)	18 j	19/02/04	08/03/04	6
Conception d'un cours test d'algorithmique	14 j	08/03/04	21/03/04	2
Evaluateur (temps passé sur une page)	3j	08/03/04	11/03/04	1
Modélisation de la base de données	8 j	11/03/04	18/03/04	4
Programmation partielle de l'algorithme	15 j	24/03/04	08/04/04	2
Rédaction du cahier des charges	21j	22/03/04	12/04/04	4
Préparation de la pré-soutenance	2j	13/04/04	14/04/04	6

Pendant la phase de production, nous avons travaillé en moyenne 5 jours par semaine, à raison de 5 heures par jour. Nous avons rédigé le rapport tout au long du projet, et le temps indiqué dans les tableaux ci-dessous correspond à sa finalisation.
La phase de test s'est rapidement révélée fastidieuse, et nous avons manqué de temps pour réaliser tous les tests pertinents.
Notre planning du cahier des charges est faussé, car nous avons comptabilisé les week-ends comme jours de travail.
Comme prévu, nous nous sommes regroupés par binômes pour réaliser toutes les tâches de notre planning. Nous les analyserons donc par groupe de travail.

Binôme 1 : Corinne Albarelli et Laetitia Joly

Tâche	***Durée prévue***	***Durée effective***
Conception du site Web	7j	7j
Programmation des robots étudiants	9j	5j

Tests	10j pour les tests et leur analyse	6j
Rédaction du rapport	Rédaction tout au long du projet + 5j pour la finalisation	6j
Documentation	-	2

Pour la programmation des robots étudiants, nous avions pensé décrire le cours sous forme d'un schéma XML : le robot devait récupérer dans le fichier XML les informations nécessaires pour avancer dans le cours. Nous avons abandonné cette idée pour une autre plus simple et plus rapide à implémenter. Ceci justifiant le fait que l'on ait mis moins de temps que prévu.
La phase de tests a été assez longue : le lancement des robots a permis d'effectuer les bons paramétrages, aussi bien au niveau de l'algorithme qu'au niveau des robots eux-mêmes. L'analyse des tests a finalement été confiée au binôme Emilie Rouch et Benoît Missonier.
En ce qui concerne la documentation, nous avions oublié de la mentionner dans le cahier des charges.

Binôme 2 : Patrice Behem et Jérôme Guillot

Tâche	***Durée prévue***	***Durée effective***
Programmation de l'algorithme	15j	15j
Intégration de l'algorithme dans le site	5j	2j
Programmation de l'application de test	-	5j
Paramétrage	2j	3j
Rédaction du rapport	Rédaction tout au long du projet + 5j pour la finalisation	5j

Dans le cahier des charges, nous n'avions pas prévu la programmation d'une application de test. Mais elle n'a posé aucune difficulté majeure, et les cinq jours que nous lui avons consacré se sont déroulés en parallèle avec les derniers jours de la programmation de l'algorithme ACO.
L'intégration de l'algorithme dans le site a pris moins de temps que prévu officiellement. Mais dans les faits, à chaque modification (ou amélioration) apportée au site ou à l'algorithme, il fallait également apporter de légères modifications pour l'intégration dans le site. Cette intégration a donc été en réalité une tâche « perpétuelle », il est donc difficile de lui attribuer une durée de travail effective.

Binôme 3 : Emilie Rouch et Benoît Missonier

La gestion des pages d'exercices ayant été omise, elle a été confiée à ce binôme.

Tâche	*Durée prévue*	*Durée effective*
Gestion des exercices	-	11j
Programmation des évaluateurs	9j	9j
Analyse des résultats	-	2j
Rédaction du rapport	Rédaction tout au long du projet + 5j pour la finalisation	6j

La durée de la gestion des exercices a été longue, car il a fallu déterminer ce que pouvait être une page d'exercices et comment la relier au cours. Ensuite il a fallu déterminer comment nous allions sélectionner les exercices en fonction du chapitre courant du cours et du niveau de difficulté des exercices. C'est donc l'élaboration de cet algorithme qui nous a pris le plus de temps. Enfin, nous avons perdu du temps à cause des nombreux changements de structure de la base de données.
Nous avons mis moins de temps que prévu pour la programmation des évaluateurs, car nous ne les avons appliqués que sur une seule préférence.
L'analyse des résultats a été effectuée en parallèle avec la rédaction du rapport, ce qui signifie que les 2 jours d'analyse des résultats sont aussi compris dans les 6 jours de rédaction du rapport.

Tâche commune

Durant la dernière semaine, après avoir rendu le rapport ainsi que les sources et la documentation, nous préparons la soutenance finale qui a lieu le 17 juin 2004.

3.2 Méthodes de travail

Lors de la phase de production, nous avons choisi de diviser le travail en trois binômes pour réaliser les trois premières tâches principales de notre projet, à savoir la programmation de l'algorithme, la réalisation du site Web, et la programmation des évaluateurs. Les trois binômes ont travaillé de manière coordonnée : la réalisation du site Web et la programmation de l'algorithme ont été effectuées en parallèle par deux équipes différentes, pour pouvoir intégrer l'algorithme au site à la fin de ces deux étapes. Toujours en parallèle, le binôme 3 a réalisé la gestion des exercices, puis les évaluateurs, afin de les intégrer respectivement dans le site Web et dans l'algorithme. Ensuite, le binôme 2 a implémenté l'application de test pour traiter les résultats des robots étudiants, pendant que le binôme 1 s'est chargé de réaliser ces robots, ainsi que les tests.

Les tâches réalisées étant liées les unes aux autres, les groupes de travail ont communiqué quotidiennement ; notamment, le binôme 2 a demandé au binôme 3 la programmation des évaluateurs dont il avait besoin.
Pendant les paramétrages de l'algorithme d'une part, et ceux des robots d'autre part, le binôme 3 a analysé les résultats obtenus par les deux autres binômes.

Chaque binôme a rédigé la partie du rapport correspondant à son travail.

De plus, nous avons organisé des réunions tous les six au moins une fois par semaine afin de se mettre d’accord sur le travail réalisé et à réaliser, et de faire les mises au point nécessaires.

Conclusion

Bilan du TER

Afin de dresser un bilan de notre TER, nous allons tout d'abord récapituler le travail effectué. Nous avons déterminé les critères entrant dans la définition du profil de l'étudiant, et ce qu'était un bon cours pour tous les étudiants, quel que soit leur profil. Nous avons ensuite mis en place l'algorithme par colonies de fourmis afin de présenter à l'étudiant un cours adapté à son profil. Nous avons créé un site Web afin que l'étudiant ait accès aux cours. Nous avons implémenté des évaluateurs afin de permettre l'évolution du profil de l'étudiant. Nous avons réalisé des robots étudiants afin de tester le bon fonctionnement de l'algorithme, ainsi qu'une application de test pour visualiser les résultats des robots sous forme graphique afin de les analyser.

Ce TER a été pour nous l'occasion d'appliquer l'algorithme par colonies de fourmis à un cas concret. Il existe beaucoup de sites d'e-learning, mais peu sont évolutifs. Ceci ajoute un intérêt supplémentaire à notre travail.

Ce TER nous a permis de réaliser une application complète, de la phase de conception à la phase de test. Nous avons pu ainsi voir les difficultés qui en découlent.

De plus, nous avons travaillé en équipe de 6, ce qui a nécessité de l'organisation et de la coordination ; cela représente une expérience très intéressante pour préparer notre avenir professionnel.

Si c'était à refaire

Nous avons perdu du temps à cause de changements répétés de la structure de la base de données. Peut-être aurait-il fallu réfléchir davantage avant de commencer l'implémentation. Mais ces changements ont correspondu à des besoins lors de l'implémentation, et nous ne pouvions pas forcément les prévoir.

Nous avons également perdu du temps avec les briques logicielles : en effet, nous n'avons pas compris dès le départ ce qu'elles permettraient de calculer. Nous avons d'ailleurs changé leur nom en « évaluateur », puisque nous ne les avions pas implémentées comme des bouts de code réutilisables.

Nous n'avons pas pu réaliser tous les tests pertinents; nous n'avons pas prévu assez de temps pour tous les effectuer, car ils se sont révélés plus fastidieux que prévu.

Références bibliographiques

Article sur l'algorithme ACO

Artificial Ant Colonies and E-learning : An Optimisation of Pedagogical Path,
Projet Fractales – INRIA,
Auteurs : Y.SEMET, Y.JAMONT, R.BIOJOUT, E.LUTTON, P.COLLET

Sites Web

Etude de l'existant

- http://www.paraschool.com/
- http://europa.eu.int/comm/education/programmes/elearning/index_fr.html
- http://fr.sun.com/formation/e-learning/wlc/
- http://www.france5.fr/emploi/former/W00216/34/
- http://www.memopage.com
- http://www.savoir.com
- http://www.unext.com

Article sur l'algorithme ACO

- http://www.epi.asso.fr/revue/articles/a0309b.htm

Téléchargement et installation d'EasyPhp 1.7 sous windows

- http://www.toutestfacile.com/phpinit.php?tef_site=php&chap=la0

Les manuels de référence

- *MySQL* : http://dev.mysql.com/doc/mysql/fr/index.html
- *PHP* : http://www.php.net/manual/fr/
- *PHPMyAdmin* : http://cvs.sourceforge.net/viewcvs.py/phpmyadmin/pma_localized_docs/fr/Documentation.fr.html?rev=1.7#setup
- *Java* : http://java.sun.com/j2se/1.4.2/docs/api

Utilisation de *PHP*

- http://www.lephpfacile.com
- http://www.phpfrance.com/
- http://www.phpinfo.net/
- http://magali.contensin.free.fr/html/php/cours_php.html
- http://www.manuelphp.com/cours/

Envoi de requêtes à partir d'un programme *JAVA*

- http://martin.nobilitas.com/java/cookies.html

Livres

PHP et MySQL en Ligne, édition Micro Application, Jean Carfantan