Projets algorithmique et complexité
M1 informatique, 2016–2017

Pr. Dominique Michelucci, Université de Bourgogne, Dijon
LE2I, UMR CNRS 5158
Dominique.Michelucci@u-bourgogne.fr

Table des matières

Introduction

Le rapport écrit doit être réalisé en Latex ; vous pouvez utiliser Lyx. Ce texte est rédigé en Latex. Le fichier latex et le fichier bib (pour la bibliographie) sont disponibles sur le site web du M1 informatique de Dijon.

Le rapport doit être écrit en français, et long de 10 à 12 pages (sans compter les sources de votre programme, la table des matières, etc). N'imprimez pas sur papier votre rapport. Envoyez moi seulement le fichier au format PDF.

Ni organigramme, ni pseudocode.

Rédigez votre rapport avec soin. La langue naturelle est le premier langage de programmation et de modèlisation, aussi un rapport confus et criblé d'erreurs d'orthographe ou de français augure-t-il mal de votre talent d'informaticien.

Le programme ispell vérifie l'orthographe ; attention, il ne gère pas du tout

les accords (il en est de même pour word).

Donnez les références bibliographiques, les adresses des sites web que vous avez utilisés.

Pensez à commenter vos sources, sans excès. Il n'est pas imposé d'utiliser ocamldoc (similaire à javadoc).

Vous créerez une archive de votre projet, contenant les sources, le makefile, les fichiers d'exemples, votre rapport, et éventuellement le fichier de votre présentation orale (vous êtes libre sur le format du support de votre présentation orale), avec la commande

```
tar cvzf VOTRENOM.tgz votrerepertoire
```

que vous enverrez par courriel, **le 17 décembre 2015 au plus tard**, à

`dmichel@u-bourgogne.fr`

L'archive peut ne pas contenir les fichiers supports de votre présentation orale. Par contre, prévoyez un fichier makefile, et une commande "make clean", "make all", et quelques programmes de démonstration, que vous appelerez demo1, demo2, etc.

La soutenance de votre projet aura lieu (probablement le jeudi et le vendredi de) la première semaine de la rentrée.

Cela va sans dire : **n'attendez pas la dernière semaine pour commencer votre projet.**

Vous trouverez des exemples de programme graphique dans

`http://ufrsciencestech.u-bourgogne.fr/master1/mi1-tc5/SOURCES/FIG`
et dans

`http://ufrsciencestech.u-bourgogne.fr/master1/mi1-tc5/SOURCES/FIG.tgz`

Dans `http://ufrsciencestech.u-bourgogne.fr/master1/mi1-tc5/SOURCES/RAY_ML_BERNSTEIN_MIGS/`, vous trouverez des fonctions pour écrire et lire des images dans des fichiers.

Voici une liste de projets possibles. **Vous pouvez proposer vos propres projets. Demandez mon accord.**

Pourquoi Ocaml est il imposé ?

- une fois que vous maîtriserez Ocaml, vous apprendrez facilement tous les autres langages de programmation.

- si cela peut vous rassurer, rien ne vous interdit de programmer d'abord votre projet avec votre langage favori.

- vous pouvez aussi utiliser le style procédural, ou orienté objet, en Ocaml.

- Vous maîtrisez (ou vous devriez maîtriser) le C, C++, Java, ou d'autres langages procéduraux. C'est donc l'occasion d'apprendre un langage vraiment différent, et une façon différente de penser et de programmer.

- Une fois que vous saurez faire, programmer en Ocaml est plus rapide, plus sûr (un programme Ocaml qui compile a plus de chances de fonctionner qu'un programme C++ qui compile...), plus simple (vous n'avez pas à gérer la mémoire avec des delete plus ou moins hasardeux), beaucoup plus concis qu'en Java ou C++ ; d'ailleurs les prouveurs actuels comme Coq sont développés en Ocaml, ainsi que des programmes critiques.

- il y a un seul compilateur de Ocaml, celui de l'INRIA (donc pas de problème d'incompatibilité).

- les programmes en Ocaml sont plus courts que les programmes en Java, en C++, etc. Java n'a pas de fonction anonyme ; pour les simuler, il faut décrire une classe ou une interface, décrire les champs et les méthodes, allouer l'instance, initialiser les champs, appeler la méthode... C'est facilement 10 fois plus long que le programme équivalent en Ocaml. De plus, une fonction Ocaml peut dynamiquement créer et rendre une fonction (une "fermeture") ; Java ne le permet pas ; C++ non plus. Java ne permet même pas de passer une fonction en paramètre.

- les programmes en Ocaml sont davantage génériques, sans avoir à utiliser la grosse artillerie (template en C++, par exemple, qui sont, ou ont été interdits dans plusieurs entreprises).

- Java n'est pas générique, Ocaml si. Il n'y a pas d'héritage multiple en Java, contrairement à Ocaml.

- à l'exécution, les programmes en Ocaml (compilés avec Ocamlopt) sont pratiquement aussi rapides que les programmes en C++, et plus rapides que les programmes Java.

- Microsoft et Dassault Systèmes font partie du consortium Ocaml. Le langage F# de Microsoft est directement inspiré de Ocaml : Ocaml est donc une excellente introduction à F#.

- Ocaml est utilisé dans l'industrie [1] : Facebook, Dassault Systèmes, CEA...

- De nombreux cours, tutoriels, livres sur Ocaml, ou Ocaml et OpenGl [?, ?, ?, ?, ?] sont disponibles sur internet ou à la BU. Voir par exemple : `http://www.freebookcentre.net/Language/Free-OCaml-Books-Download.html`

Le plagiat est un délit

Vous pouvez utiliser des librairies ou des programmes que vous n'avez pas écrits : éditeur de texte, compilateur, préprocesseur, ocamllex, ocamlyacc, ocamlimages, GSL, lablgl, lablglut, Makefile standard dans la documentation d'ocaml, programmes réalisés lors de vos TPs... Vous n'avez pas à les mentionner dans votre rapport.

Vous pouvez récupérer quelques programmes sur des forums ou sur internet (par exemple, pour lire ou écrire dans un fichier, pour utiliser les threads, ...), mais vous devez mentionner obligatoirement tous vos emprunts (via la commande `\cite{}` de Latex, et le programme bibtex) pour ne pas être soupçonné de plagiat.

Par exemple :

Ce document, que vous êtes en train de lire, est une simple mise à jour du document de l'an passé, qui décrivait les projets pour l'année universitaire 2014–2015.

Cet avertissement contre le plagiat vaut pour les sources de vos programmes, mais aussi pour votre rapport (votre enseignant sait lui aussi se servir d'un moteur de recherche).

La notation du projet

Un projet dont le programme ne fonctionne pas ne peut pas obtenir la moyenne.

Un projet dont le rapport est mauvais ne peut pas obtenir la moyenne.

En cas de plagiat, la note est 0.

1. Voir `https://ocaml.org/learn/companies.html`.

1 Eternity II. 1 étudiant. Ocaml.

Vous trouverez sur Internet la description du jeu Eternity II. Vous générerez des puzzles aléatoires, vous en mélangerez les pièces, et vous résoudrez par une méthode de recherche arborescente avec retour en arrière (*backtrack*). Votre programme doit pouvoir résoudre des puzzles de taille 12 par 12, avec une dizaine de couleurs.

Dernière nouvelle : voir dans le répertoire UTILITES les fichiers relatifs à Eternity.

2 Eternity II et SAT. 2 étudiants. Ocaml.

Vous trouverez sur Internet la description du jeu Eternity II. Vous récupérerez sur internet un programme de satisfaction de contraintes booléennes (le problème s'appelle SAT, ou 3-SAT ; des solveurs sont minisat ou picosat). Vous chercherez sur internet "eternity II SAT" pour trouver des articles présentant comment formuler le puzzle d'Eternity II comme un problème de satisfaction de contraintes booléennes ([**?**, **?**] par exemple). Vous générerez des puzzles aléatoires du jeu Eternity II. Vous générerez le problème de contraintes booléennes correspondant, que vous sauverez dans un fichier ; vous appelerez le solveur de contraintes booléennes ; vous lirez le fichier solution. Vous testerez quelles tailles de puzzle sont solubles en un temps raisonnable (moins d'une 1/2 heure). Vous pouvez essayer plusieurs représentations du problème sous forme de contraintes booléennes, ainsi que plusieurs solveurs booléens.

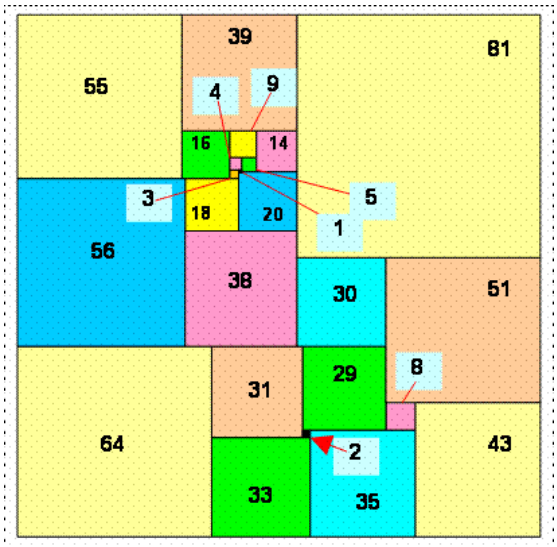
Dernière nouvelle : il existe un logiciel en Ocaml qui résout le problème SAT : SAT-MICRO, "petit mais costaud", qui est dû à Sylvain Conchon, Johannes Kanig, Stéphane Lescuyer. Il est téléchargeable sur internet. Voir aussi les fichiers "conchon*" dans le répertoire UTILITES.

Dernière nouvelle : voir dans le répertoire UTILITES les fichiers relatifs à Eternity.

3 Pavage en carrés. 2 étudiants. Ocaml.

Soit un carré de coté n unités. Pour certaines valeurs de n, il est possible de paver le carré par des carrés plus petits, de côtés entiers et tous différents. Vous chercherez d'abord si n^2 peut s'écrire comme une somme de carrés d'entiers tous distincts. On sait que ce problème ne peut pas se généraliser en 3D et au delà.

Voir `http://villemin.gerard.free.fr/Pavage/CarrParf.htm`, d'où vient cette image :



4 Solitaire. 1 étudiant. Ocaml

Un carré de 2×2 cases est enlevé aux quatre coins d'un damier 7×7. Il reste donc $72 - 4 \times 4 = 33$ cases au damier. La case centrale est vide ; les 32 autres cases sont occupées par 32 pions, à raison d'un pion par case. Le but du jeu est d'enlever tous les pions, sauf un. Vous avez le droit de déplacer un pion vers une case vide, en le faisant sauter au dessus d'un autre pion que vous devez enlever ; les 3 cases concernées sont consécutives sur une rangée ou une colonne du plateau de jeu de 33 cases.

Attention : il y a beaucoup de configurations possibles, si bien qu'il faut vraisemblablement adapter l'algorithme de recherche standard (*backtrack*) ; par exemple, il peut être nécessaire d'encoder le damier sur un entier (mais il y a 33 cases). Vous pouvez aussi utiliser des modules bitvector, disponibles via internet.

Vous essaierez ensuite d'agrandir le damier ; jusqu'où pouvez vous aller ?

5 Calcul de π. 1 étudiant. Ocaml

Empruntez le livre de Jörg Arndt et Christoph Haenel : "A la poursuite de Pi", éditions Vuibert, à la BU (cote 510/1232). Programmez 2 à 3 algorithmes parmi les 9 du livre. Il y a aussi un livre de Jean-Paul Delahaye à la BU, me semble-t-il : "Le fascinant nombre Pi". Vous téléchargerez les décimales de π pour vérifier vos résultats. Il faut utiliser le module Nums.cma. Piège : utilisez

bien `=/`, `</`, `<=/` etc !

Attention : certaines formules sont fausses... Testez les d'abord en arithmétique flottante !

Remarque : il faut utiliser la librairie nums.cma de ocaml.

```
$ ocaml nums.cma
        OCaml version 4.01.0

# open Num;;
# let printnum n = Format.printf "%s" (string_of_num n) ;;
val printnum : Num.num -> unit = <fun>
# #install_printer printnum;;
#  Int 2 // Int 4;;
- : Num.num = 1/2
# let rec facto n = if n  <= 1 then Int 1 else (Int n) */ (facto (n-1));;
val facto : int -> Num.num = <fun>
# facto 40;;
- : Num.num = 815915283247897734345611269596115894272000000000
```

6 Taquin. 2 étudiants. Ocaml

Faire des mouvements aléatoires sur un taquin. Puis résolvez le par programme (et sans tricher). Jusqu'à quelle taille de taquin pouvez vous aller avant que votre programme ne devienne trop lent ? Site web : outre wikipedia,

`http://maths.amatheurs.fr/index.php?page=taquin`

Dernière nouvelle : plusieurs méthodes de résolution du taquin sont publiées sur internet.

Remarque : la librairie graphics.cma de Ocaml est largement suffisante.

7 Cube de Rubik. 2 étudiants. Ocaml

Appliquez des rotations aléatoires sur le cube de Rubik. Puis remontez le.
Utilisez une autre méthode que celle des 3 étages.

Vous utiliserez lablgl ou lablglut.

Vous trouverez des exemples de programme graphique dans `http://ufrsciencestech.u-bourgogne.fr/master1/mi1-tc5/SOURCES/FIG` et dans `http://ufrsciencestech.u-bourgogne.fr/master1/mi1-tc5/SOURCES/FIG.tgz`

8 Sudoku. 1 étudiant. Ocaml

Le programme de DM sur le site convertit le problème du Sudoku en un problème de coloriage de graphe. Vous pouvez faire mieux ! Vous utiliserez le fait que le 1 (ou tout autre numéro de 1 à 9) d'une ligne, d'une colonne, ou d'un bloc de 3^2 cases doit bien se trouver quelque part, et que parfois il n'y a plus qu'une seule case possible (ou aucune case possible, auquel cas la branche courante dans l'arbre de recherche est une impasse). Ce principe permet le plus souvent d'éliminer tout retour en arrière. Votre programme expliquera son raisonnement, en français.

Votre programme doit toujours être capable d'effectuer des tentatives et des retours en arrière, pour les sudokus les plus difficiles.

Vous comparerez votre programme (en mesurant le nombre de retour en arrière effectués) avec celui de DM disponible sur le site.

Vous testerez des sudokus de taille arbitraire (les nombres utilisés allant de 1 à n^2). Jusqu'où pouvez vous aller ?

Dans une seconde partie, vous essaierez de générer des jeux de Sudoku, avec juste les données suffisantes (un minimum de cases remplies) pour que la solution soit unique.

9 Recherche d'une chaîne de caractères. 1 étudiant. Ocaml

Programmer toutes les méthodes suivantes de recherche d'une chaine de caractères dans un fichier : la méthode naïve, Rabin-Karp (pas besoin d'utiliser la librairie nums !), par automate fini déterministe, Knuth-Morris-Pratt. Toutes ces méthodes sont présentées dans [**?**].

10 Algorithme de Johnson. 1 étudiant. Ocaml

Programmez l'algorithme de Johnson [**?**], qui calcule tous les plus courts chemins dans un graphe sans circuit négatif. Tracez la courbe donnant le temps de calcul en fonction de la taille du graphe, pour prouver que votre implantation est bien correcte.

11 Flot maximum. 1 étudiant. Ocaml

Programmez la recherche du flot maximum dans un graphe. Utilisez la méthode de Ford-Fulkerson modifiée par Karp, ou une méthode de préflots. Ces méthodes sont présentées dans [**?**]. Vous pouvez utiliser le logiciel libre dotty pour visualiser le graphe. On ne demande pas d'interface interactive pour entrer ou modifier le graphe. Prévoyez un format de fichier commode pour décrire les données du problème (le graphe et les capacités minimales et maximales des arcs) ; d'ailleurs, cela peut être un programme Ocaml. Prévoyez aussi une interface (pas au sens IHM !) par appel de fonctions : flotmax legraphe, et les fonctions nécessaires pour construire un graphe et les capacités des arcs.

12 Couplage optimal. 1 étudiant. Ocaml

n étudiants doivent se répartir n projets dans le module Algorithmique et Complexité. Chaque étudiant classe les projets en $1, 2..., n$, par priorité décroissante. Trouver une correspondance un-un (biunivoque, ou bijective) qui minimise le mécontentement des étudiants est un problème de couplage optimal. Programmez l'algorithme hongrois [**?**], ou une autre méthode en temps polynomial. Toute méthode polynomiale calculant le flot de coût minimum convient. Tracez la courbe du temps d'exécution en fonction de n.

13 Triangulation de surfaces implicites $f(x, y, z) = 0$. 3 étudiants. Ocaml

Vous généraliserez au 3D la méthode vue en 2D pour dessiner des courbes implicites données par une équation $f(x, y) = 0$. Pour chaque cube (voxel) élémentaire traversé par la surface d'équation $f(x, y, z) = 0$, vous partitionnerez le cube en 5 (ou 6) tétraèdres, et vous approcherez l'intersection de la surface avec un tétraèdre par un triangle ou un quadrilatère. Vous utiliserez une arithmétique d'intervalles.

Ocaml fournit un lex et un yacc : Ocamllex, Ocamlyacc. Vous pouvez les utiliser pour analyser des fichiers de description de surface. Mais l'utilisation de ces outils (Ocamllex, Ocamlyacc) n'est pas prioritaire : vous pouvez vous contenter de redéfinir les opérations $+$, $-$, $\times$, etc pour des DAG comme vu en TP. Pour la visualisation d'un ensemble de faces, vous pouvez utiliser, ou modifier, les programmes de DM dans
`http://ufrsciencestech.u-bourgogne.fr/master1/mi1-tc5/SOURCES/FIG` et dans `http://ufrsciencestech.u-bourgogne.fr/master1/mi1-tc5/SOURCES/FIG.tgz`

Voir aussi : `http://www.opengl.org/resources/faq/technical/miscellaneous.htm`

14 Triangulation de surfaces algébriques implicites. 3 étudiants. Ocaml

C'est le même projet que le précédent, mais les surfaces sont algébriques et vous utiliserez les bases de Bernstein et la méthode de Paul Faget de Casteljau, qui est bien expliquée dans Wikipedia, dans le cas des fonctions à 1 et 2 variables ; l'extension au cas de 3 variables est immédiate. Vous utiliserez cet algorithme pour calculer les encadrements de $f(x, y, z)$, avec x, y, z dans une boite donnée.

Pour la partie graphique, vous pouvez vous inspirer ou réutiliser les sources dans

`http://math.u-bourgogne.fr/michelucci/OCAML/FIG/` ou `http://math.u-bourgogne.fr/michelucci/OCAML/FIG.tgz`.

15 Dessin de formes 2D, par intervalles. Ocaml. 1 étudiant.

Une forme 2D est soit
- le carré unité $[0, 1]^2$,
- le cercle unité, les points (x, y) tels que $1 - x^2 - y^2 - 1 \geq 0$,
- un demi plan (a, b, c) : les points (x, y) tels que $ax + by + c \geq 0$
- l'union de deux formes $(a \cap b)$,
- l'intersection de deux formes $(a \cup b)$,
- la différence de deux formes $(a - b)$
- la translation de vecteur (u, v) d'une forme,
- la rotation de θ degrés autour de l'origine d'une forme,
- l'affinité de facteurs a, b d'une forme $(x' = ax, y = by)$,
- une primitive décrite par une expression $f(x, y) = 0$; f est décrite par un DAG ;
- tout (si c'est commode d'avoir une telle forme),
- rien (si c'est commode d'avoir une telle forme).

Vous généraliserez la méthode vue en TP (celle qui trace des courbes données par une équation $f(x, y) = 0$, en utilisant une arithmétique d'intervalles) à ces formes. Vous permettrez aussi à l'utilisateur de spécifier la couleur, et éventuellement la profondeur des formes. graphics.cma suffit.

16 Percolation. 1 étudiant. Ocaml

Une grille 2D est partitionnée en $n \times n$ carrés. Chaque carré est blanc avec une probabilité p, ou noir avec une probabilité $1 - p$. Il y a percolation quand il existe un chemin de cases noires d'une case dans la ligne du haut de la grille vers une case de la ligne en bas de la grille. Faites un programme pour mesurer empiriquement la probabilité de percolation en fonction de p et de n. Que constatez vous ? On peut se poser d'autres questions, telles que la probabilité d'un chemin blanc entre la colonne gauche et la colonne droite, l'influence de la connexité (un carré a-t-il 4 ou 8 voisins ?). Des applications de la théorie de la percolation sont l'analyse des risques de la propagation des incendies, épidémies, rumeurs, etc.

17 Programmation linéaire. 1 étudiant. Ocaml

Empruntez le livre "Recipes in C" [**?**], et traduisez le programme du simplexe qu'il contient en Ocaml. Vos indices dans les tableaux commenceront à 0 (ils commencent à 1 dans le livre).

18 Programmation linéaire. 1 étudiant. Ocaml

Programmez la méthode du simplexe en 2 passes qui est décrite dans la dernière version du livre "Introduction à l'algorithmique", empruntable à la BU. Attention, il vous faudra corriger une petite erreur.

19 Programmation linéaire avec gradients conjugués. Ocaml. 2 ou 3 étudiants.

Encadrants : Jean-Marc Cane, Arnaud Kubicki, Dominique Michelucci.

Un polytope (polyèdre convexe) est décrit en n dimensions (n grand : 500, ou 1000) l'intersection d'une boite (parallèle aux axes) et d'un sous espace affine (de dimension inférieure à $n-1$), décrit par $Ax = B$, où A une matrice donnée m, n, avec $1 < m < n$. une méthode de gradients conjugués (lire : Numerical Recipes in C, disponible sur internet, n'importe quel livre d'analyse numérique empruntable à la BU ; attention : l'article sur les gradients conjugués est trompeur (en sept. 2011) ; il faut lui préférer l'article sur la méthode de Broyden, Fletcher, Goldfarb, Shanno (BFGS method)) pour trouver un point du polytope, ou si le polytope est vide, pour trouver les deux points les plus proches, l'un sur la boite, l'autre sur le sous espace affine. Soit $a(x)$ la distance du point

à l'espace affine $Ax = B$ (il faut savoir projeter un point x sur un sous espace affine ; c'est de l'algèbre linéaire ; demandez les détails à D. Michelucci ; vous pouvez aussi trouver ça sur internet ou dans un livre d'algèbre linéaire) ; soit $b(x)$ la distance de x à la boite ; la méthode des gradients conjugués permet de trouver le minimum de $F(x) = (a(x) - b(x))^2 + a(x) + b(x)$. On peut aussi utiliser $F(x) = a(x)^2 + b(x)^2$.

Pour vos premiers essais, vous considèrerez des exemples 2D : la boite est un carré, l'espace affine est une droite qui coupe, ou non, le carré. La librairie graphics.cma est suffisante. Ensuite vous considèrerez des exemples en plus haute dimension ; vous visualiserez la courbe (décroissante) de F en fonction des itérations. Vous comparerez plusieurs variantes, que vous trouverez sur `http://en.wikipedia.org/wiki/BFGS_method` ou ailleurs, par exemple la formule de Davidon, Fletcher, et Powell
`http://en.wikipedia.org/wiki/Davidon-Fletcher-Powell_formula`

Vous pouvez stopper la méthode dès que vous avez trouvé un point à l'intérieur du convexe (donc le convexe est non vide), ou que vous avez trouvé un hyperplan contenant le sous espace affine et ne coupant pas la boite (donc le convexe est vide).

La BU fournit les 2 tomes du livre [**?**] : "Programmation Fonctionnelle appliquée aux calculs scientifiques Tome I, Tome II" de Couturier et Jean-Baptiste, édition Cépaduè, 2007.

Projet "mathématique", bien sûr...

20 Algèbre linéaire. 1 étudiant. Ocaml

Empruntez [**?**] à la BU ; il est aussi disponible sur internet. Vous traduirez en Ocaml les fonctions effectuant les décompositions LUP et SVD, le calcul de l'inverse, la résolution de systèmes linéaires. Vos tableaux doivent avoir 0 en premier indice (et non 1, comme dans [**?**]). Votre module Ocaml doit pouvoir être utilisé facilement par un programmeur en Ocaml.

21 Multiplication de Strassen. 1 étudiant. Ocaml

La méthode de Strassen permet de multiplier des matrices carrées n par n en moins de temps que $O(n^3)$. Vous programmerez cette mltiplication, ainsi que l'inversion basée sur la mutiplication rapide de Strassen [**?**]. Vous comparerez les vitesses de la multiplication de Strassen et de la multiplication usuelle pour différentes valeurs de n. Vous testerez aussi la précision du calcul de l'inverse (la différence entre MM^{-1} et l'identité). Remarque : votre programme ne doit pas supposer que n est une puissance de 2. Dans votre rapport, vous expliquerez

pourquoi cette méthode ne peut pas être utilisée pour le calcul des plus courts chemins.

22 Interpolation. 1 ou 2 étudiants. Ocaml

Vous calculerez la courbe algébrique $f(x,y) = \sum_i \sum_j a_{ij} x^i y^j = 0$ $(i+j \leq d)$ de degré d donné qui passe, ou approche, un ensemble donnés de points dans le plan. Vous utiliserez une résolution aux moindres carrés [**?**, **?**]. Vous afficherez la courbe et les points. Les points peuvent être désignés avec la souris. Vous pouvez aussi générer des points sur une courbe connue (cercle, ellipse), pour vérifier que vous retrouviez bien la courbe en question.

Pour 2 étudiants : vous chercherez ensuite un autre type d'interpolation, par exemple par noyau gaussien (gaussian kernel). Mots clefs pour la recherche sur internet : gaussian kernel, radial basis function.

23 Compression de fichiers. 1 étudiant. Ocaml

Programmer la méthode de compression et de décompression de fichiers de Huffman. Voir `http://en.wikipedia.org/wiki/Huffman_coding`

24 Méthode de Dijkstra. 1 étudiant. Ocaml

Programmez la recherche du plus court chemin par la méthode de Dijkstra, en testant plusieurs structures de données pour la queue de priorité : un tas (heap), tel que les sommets connaissent leur position dans le tas (le tas est mis à jour quand la distance du sommet à la source diminue), un tas de Fibonacci [**?**], un tas binomial [**?**]. Vous pouvez récupérez sur internet des implantations des tas de Fibonacci et des tas binomiaux ; par contre, vous programmerez le tas (heap).

Vous implanterez un module avec foncteur pour chacune des structures de données. Il faut pouvoir passer en paramètre un module qui fournit une fonction de comparaison (et peut-être d'autres petites choses). Pour l'interface (au sens de signature), vous pouvez vous inspirer des modules Set, Hashtabl de Ocaml.

25 Stéganographie. 1 ou 2 étudiants. Ocaml

Vous utiliserez le logiciel convert, ou xv, sous linux pour convertir les images dans des formats de fichiers facilement lisibles en Ocaml. Vous pouvez utilisez

la librairie camlimages.

Dans une image RVB, il est souvent possible de modifier, sans effet visible, le bit de poids faible de l'octet des composantes rouge, verte, bleue. Une méthode de stéganographie consiste à sacrifier ces bits et à les remplacer par ceux d'un texte encrypté (ou d'une image binaire encryptée) par la méthode RSA vue en cours de cryptographie. Ocaml fournit une arithmétique sur des entiers (ou des rationnels) de longueur arbitraire : nums.cma ; vous l'utiliserez. Ecrire le programme d'encryptage et de décryptage. Les clefs publiques et privées seront contenues dans des fichiers.

Vous aurez besoin de calculer des grands entiers premiers p et q. La clef publique est le produit $n = pq$. La clef secrète est le couple (p, q). Vous utiliserez le test probabiliste de primalité donné en [**?**].

Erreur à ne pas commettre : encrypter chaque bit (ou chaque octet) séparément... Si le message à encrypter est court, il faut le compléter avec du bruit (des valeurs aléatoires).

26 FFT et multiplication de polynômes. 1 étudiant. Ocaml

Lire le chapitre consacré dans [**?**], et programmer cette méthode. Programmer 2 variantes : les coefficients sont des nombres flottants, et les coefficients appartiennent à un corps fini, les calculs sont effectués modulo un nombre premier. Dans ce dernier cas, vous pouvez utiliser la librairie num de Ocaml.

27 Arithmétique sur de grands entiers. 2 étudiants. Ocaml

Programmer une arithmétique $(+, -, \times, :, \mathrm{mod})$ sur de grands entiers et de grands rationnels (sans utiliser nums.cma). Programmer aussi le pgcd. Pour la multiplication, utiliser la méthode de Karatsuba. Tester avec la fonction factorielle.

28 Algorithmique et arithmétique. Ocaml. 2 étudiants

Vous utiliserez la librairie nums.cma, et vous programmerez le test de primalité de Miller-Rabin [**?**], la méthode de factorisation rho de Pollard [**?**], la recherche de la racine carrée modulo p (livre de Naudin et Quitté : algorithmique géométrique, à la bibliothèque : 518.4–1099), et le critère d'Euler.

29 Transformée de Fourier et multiplication de grands entiers. Ocaml. 2 étudiants

En utilisant la librairie nums.cma, programmer la multiplication de grands entiers avec la transformée rapide de Fourier (Introduction à l'algorithmique, de Cormen, Leiserson, Rivest) et la multiplication par la méthode de Karatsuba. Comparer les performances des deux méthodes. Testez en calculant la factorielle de 1000 ou 10000. Ces notions sont très utilisées en cryptographie (de nombreux livres y sont consacrés).

30 Coloration de graphes. 1 étudiant. Ocaml

Programmer et optimiser votre propre méthode de coloration de graphe. Testez la sur différents graphes ; par exemple, sur divers graphes aléatoires (par exemple, toute arête a une probabilité p d'exister ; évaluer le nombre minimal de couleurs, et les performances de votre méthode, en fonction de p et du nombre n de sommets), de complexité n croissante.

Vous testerez aussi votre méthode sur des grilles de Sudoku. Il faut donc écrire une fonction pour traduire une grille de Sudoku en problème de coloriage de graphes.

Applications : sudoku ou autres casse-têtes, allocation de ressources, par exemple allocation de registres en compilation.

31 Factorisation d'entiers. 1 ou 2 étudiants. Ocaml

Réduire le problème de la factorisation d'un entier à un problème SAT (satisfiabilité d'une formule logique). Pour cette réduction, il faut multiplier, en base 2, $x = x_k \dots x_0$ par $y = y_k \dots y_0$, pour que $xy = n$, où n est le nombre à factoriser, et où les x_i et y_i sont des booléens inconnus. Vous avez le droit d'utiliser des variables auxilliaires (des retenues pour les additions, par exemple), pour trouver la formule logique correspondante. Bien sûr, tentez d'utiliser un nombre raisonnable d'inconnues, polynomial en $\log n$. Vous utiliserez un solveur de contraintes booléennes tels que minisat ou picsat, téléchargeable gratuitement sur internet. Tous ces solveurs utilisent le même format de fichier. Il doit vous être possible de factoriser un nombre inférieur à 1000.

Dernière nouvelle : il existe un logiciel en Ocaml qui résout le problème SAT : SAT-MICRO, "petit mais costaud", qui est dû à Sylvain Conchon, Johannes Kanig, Stéphane Lescuyer. Il est téléchargeable sur internet. Les articles sont dans le répertoire UTILITES.

32 Métamorphose (*morphing*) entre 2 photos de visages. 2 étudiants. Ocaml

Faites votre propre logiciel de métamorphose *"morphing"*.

Votre programme permettra à l'utilisateur d'ajuster interactivement des sommets caractéristiques sur chaque visage (les coins des yeux, des sourcils, de la bouche, etc). Eventuellement, votre logiciel essaiera de retrouver automatiquement ces ajustements. Ces sommets sont ceux d'un maillage constitués de triangles. Après prétraitement des deux visages, vous déformerez le premier maillage pour qu'il coïncide avec le deuxième ; par exemple si le sommet P est en P_1 sur le premier maillage et en P_2 sur le deuxième, son mouvement sera décrit par $P(t) = P_1 + (1-t)P_2$. Bien sûr, il faut aussi interpoler les pixels à l'intérieur de chaque triangle (ou quadrilatère).

Le site `http://math.u-bourgogne.fr/michelucci/OCAML/RAY_ML_BERNSTEIN_MIGS_4/` contient des fichiers de manipulation d'images : image.ml, diaporama.ml, que vous pouvez utiliser et modifier. Vous pouvez aussi utiliser la librairie camlimages, disponible sur la toile.

Vous utiliserez le logiciel convert, ou xv, sous linux pour convertir les images dans des formats de fichiers facilement lisibles. Vous pouvez utilisez la librairie camlimages.

33 Puissance 4. 1 étudiant. Ocaml

Programmer le jeu Puissance 4 ; l'utilisateur doit pouvoir jouer contre l'ordinateur ; La librairie graphics.cma est suffisante. Programmer un minimax avec élagage alpha-beta, en utilisant le programme fourni dans [**?**, **?**], disponible librement sur internet (ou à la BU). Soignez l'interface graphique.

34 Othello, Morpion, Puissance 4, ou autre jeu, et optimisation. 2 ou 3 étudiants. Ocaml

Vous utiliserez le minimax avec élagage alpha-beta fourni dans [**?**, **?**], disponible librement sur internet (ou à la BU). Vous soignerez l'interface graphique. Vous optimiserez les paramètres de la fonction évaluant la qualité d'une configuration : pour cela, vous ferez jouer entre eux des programmes avec différents paramètres, et vous sélectionnerez les programmes les meilleurs. Vous pouvez lire quelques articles sur les algorithmes évolutionnaires, inspirés de la théorie Darwinienne de l'évolution (swarm optimization, genetic algorithms, metaheuristics).

35 Morpion. 1 étudiant. Ocaml

Programmer le jeu du morpion sur une grille de 8×8 sommets. L'utilisateur doit pouvoir jouer contre l'ordinateur ; programmer un minimax avec élagage alpha-beta. Le principe est donné dans wikipedia. Un minimax est fourni dans [**?**, **?**], disponible librement sur internet (ou à la BU). Soignez l'interface graphique. La librairie graphics.cma est suffisante.

36 Complétion de Knuth-Bendix. 1 étudiant. Ocaml

Programmer l'algorithme de complétion de Knuth-Bendix, uniquement sur les mots (et pas des arbres, des expressions). *Aucune interface graphique n'est demandée.* La page wikipedia donne une bonne introduction, qui devrait être suffisante.

Voir `http://www.vermi.fr/blog/files/RapportKB.pdf`, wikipedia, etc. Vous utiliserez l'exemple de wikipedia.

Vous générerez ensuite tous les mots non réductibles, d'une longueur inférieure à un seuil donné.

Des exemples de groupe : les rotations et symétries du cube, du tétraèdre, du dodécaèdre régulier ; le groupe symétrique (le groupe des permutations) de n éléments.

Remarque : les "lettres" des mots peuvent être des chaines telles que `x'` ou `x''`.

Remarque : il y a plus à comprendre qu'à programmer.

37 Géométrie dynamique. 3 étudiants. Ocaml

Téléchargez GeoGebra (ou un autre lgiciel de géométrie dynamique), jouez un peu avec, avec les théorèmes de Pappus, Desargues, Pascal (soient 6 points sur un cercle ou une autre conique ; alors les cotés opposés se coupent selon 3 points alignés ; cette propriété reste vraie lorque l'on modifie la position d'un point sur le cercle). Ensuite, programmez votre propre logiciel de géométrie dynamique, et utilisez le pour prouver une dizaine de théorèmes géométriques, et tracer quelques lieux géométriques (ex : la position d'un point d'intersection quand un point "de base" se déplace le long d'une droite ou d'un cercle).

Il vous faudra représenter par un arbre les relations de dépendances entre les éléments géométriques (points, droites, cercles). Les éléments aux feuilles de l'arbre sont des éléments de base, qui ne dépendent d'aucun autre. Les noeuds

représentent les droites par 2 points (2 fils du noeud), les cercles par 3 points (3 fils du noeud), les points d'intersection entre 2 éléments fils du noeud, etc.

Un élément de base peuvent être sélectionnés et déplacés ; les éléments de l'arbre sont mis à jour.

Un embryon de logiciel de géométrie dynamique se trouve sur : `http://ufrsciencestech.u-bourgogne.fr/master1/mi1-tc5/SOURCES/GEOMETRIE_DYNAMIQUE/`

38 Isomorphisme de graphes. 1 étudiant. Ocaml

Ecrire une fonction qui décide si 2 graphes non orientés sont isomorphes. De plus la fonction retourne une liste des isomorphismes (il peut y en avoir plusieurs). Un isomorphisme est un appariement des sommets des deux graphes. Vous testerez pour de grands graphes réguliers (hypercube, hypertore).

39 Calcul des nombres de Ramsey. 2 étudiants. Ocaml

K_n est le graphe complet (avec toutes les arêtes possibles) avec n sommets. Quel que soit le coloriage en 2 couleurs des arêtes de K_6, il contient un K_3 monochrome. Par contre il y a des coloriages en 2 couleurs des arêtes de K_5 où aucun K_3 n'est monochrome. On dit que le nombre de Ramsey de 3 est 6 : un K_3 monochrome devient inévitable à partir K_6. Votre programme calculera les nombres de Ramsey de $3, 4, \ldots n$. Jusqu'où pouvez vous aller ? Vous utiliserez un programme de "backtrack" pour décider si les arêtes de K_n sont coloriables en 2 couleurs sans qu'il y ait de K_r. Attention : ce projet requiert une tournure d'esprit très mathématique.

40 Prolongation de suites. Ocaml. 1 étudiant

Une suite entière $a_i, i \in \mathbb{N}$ est donnée par ses 10 ou 20 premiers termes. On l'encode par une série formelle $\sum_i a_i x^i$. Ecrire un algorithme qui trouve la loi de formation de la série, et qui calcule les termes de rang quelconque.

Par exemple, à partir des premiers termes de la suite de Fibonacci : $1, 1, 2, 3, 5, 8, 13, 21$, cette heuristique doit trouver la relation $\psi(x)(1 - x - x^2) = 1$. Cette dernière série $\phi(x)$ a pour coefficient du monome x^i le i ème terme de la suite de Fibo-

nacci. Remarquez que :

$$\begin{array}{rcccccccc} \phi(x) & = & 1x & +1x^2 & +2x^3 & +3x^4 & +5x^5 & +8x^6 & +\dots \\ x\phi(x) & = & & 1x^2 & +1x^3 & +2x^4 & +3x^5 & +5x^6 & +\dots \\ x^2\phi(x) & = & & & 1x^3 & +1x^4 & +2x^5 & +3x^6 & +\dots \\ (1-x-x^2)\phi(x) & = & 1 & & & & & & \end{array}$$

Il faut donc deviner une relation linéaire entre les vecteurs colonnes des coefficients de la suite et ses décalages successifs. Ici il y a la même relation dans tous les vecteurs colonnes : $(1,0,0)$, $(1,1,0)$, $(2,1,1)$, etc. 3 est le plus petit nombre pour lequel il y a une telle relation, et vous essaierez dans l'ordre $k = 1, 2, 3, \dots$ jusqu'à trouver la valeur du degré.

Plus généralement, votre méthode doit trouver une relation $\psi(x)q(x) = p(x)$ où p et q sont des polynômes en x. Ensuite, en utilisant la méthode d'exposant rapide (cf IA), vous calculerez les termes quelconques de la suite.

Cette méthode fonctionne sur les suites polynomiales, exponentielles, périodiques (la période doit être plus petite que le nombre de termes disponibles, bien évidemment).

Mots clefs pour une recherche sur internet : Sloane, inverseur de Simon Plouffe...

41 Résoudre $x^2 + Ny^2 = P$ dans $\mathbb{N}$. Ocaml. 2 étudiants.

Ecrire un programme Ocaml pour résoudre $x^2 + Ny^2 = P$ dans $\mathbb{N}$ avec P un nombre premier. De plus $0 < N < P$. La méthode nécessite le calcul des racines carrées modulo P et la réduction des bases de réseaux de $\mathbb{Z}^2$ par l'algorithme de Gauss.

Un cas particulier célèbre est $N = 1$, ce qui donne l'équation $x^2 + y^2 = P$. Selon le théorème de Noël dû à Fermat, tout nombre premier P est somme de 2 carrés ssi $P = 2$ ou si P est égal à 1 modulo 4 (25-12-1640, lettre de Fermat à Mersenne). Les preuves de ce théorème ne sont pas toujours constructives, ou alors trop lentes, comme la méthode triviale (essayer toutes les valeurs possibles, elles sont en nombre fini).

Exemple : pour le petit nombre de Mersenne $P = 2^{31} - 1$, et $N = 6$, la solution est $(x, y) = (44029, 5901)$. Quelle est elle pour $N = 3, P = 2^{521} - 1$?

Algorithme L'équation est d'abord résolue dans $\mathbb{Z}/P\mathbb{Z}$, autrement dit on résout : $x^2 + Ny^2 = 0 \bmod P \Rightarrow x^2 = -Ny^2 \bmod P$. Si N n'est pas un carré dans $\mathbb{Z}/P\mathbb{Z}$, il n'y a pas de solution dans le corps fini $\mathbb{Z}/P\mathbb{Z}$, et a fortiori il

n'y en a pas dans $\mathbb{N}$. Supposons que R soit une racine carrée de $-N$, modulo P. Alors les couples (x, y) solutions modulo P forment deux droites discrètes d'équation : $x = Ry$ et $x = -Ry$. Considèrons la droite $x = Ry$, contenant les points : $(0, 0), (R \bmod P, 1), (2R \bmod P, 2) \ldots, (kR \bmod P, k)$ pour $k \in \mathbb{Z}/P\mathbb{Z}$. La figure montre la droite pour $P = 13, N = 1, R = 5$. Ces points forment une partie d'un réseau de $\mathbb{Z}^2$. Ce réseau est obtenu en prolongeant le dessin hors du carré $[0, P]^2$. Chacun des points (a, b) de ce réseau est par construction tel que $a^2+Nb^2 = mP$, avec m entier positif ou nul (car N est positif). On sait résoudre l'équation dans $\mathbb{N}$ quand on sait trouver (a, b) tel que $m = 1$. Il suffit donc de trouver le sommet du réseau le plus proche de l'origine, et différent de l'origine. Soit ce point est tel que $m = 1$ et alors ce point est la solution cherchée ; soit ce point est tel que $m > 1$ et alors l'équation n'a pas de solution dans $\mathbb{N}$.

Or trouver le point d'un réseau (on dit aussi lattice, ou treillis, ou module) de $\mathbb{Z}^2$ le plus proche de l'origine est trivial quand on connait une base minimale (U, V) du réseau ; alors $U = (U_x, U_y)$ est le vecteur le plus court, donc (U_x, U_y) est le sommet le plus proche de l'origine, et $V = (V_x, V_y)$ est le plus court des vecteurs indépendants de U. Cette base est unique (aux signes près de U_x, U_y, V_x, V_y). Elle est calculée rapidement par une méthode due à Gauss. Il ne reste ensuite plus qu'à vérifier que $U_x^2 + NU_y^2 = P$ (et pas un multiple de P).

Résumons l'algorithme. Trouver R une racine carrée de $-N$ modulo P. Si $-N$ n'est pas carré, alors l'équation est sans solution dans $\mathbb{N}$. Calculer une base (pas forcément la plus courte) du réseau ; un premier vecteur de la base est $(R, 1)$; le second est le premier vecteur $(kR \bmod P, k)$ différent de (kR, k), pour $k \in \mathbb{N}$. Réduire cette base par la méthode de Gauss. Soit (U, V) la base minimale avec $||U|| \leq ||V||$. Si l'équat ion a une solution, c'est (U_x, U_y).

Réseaux de $\mathbb{Z}^2$, bases minimales, réduction de Gauss Un réseau de $\mathbb{Z}^2$ est l'ensemble des combinaisons linéaires à coefficients entiers relatifs d'une base de 2 vecteurs de $\mathbb{Z}^2$. Deux bases sont équivalentes ssi elles engendrent le même réseau. Pour les réseaux 2D, il existe une base minimale, contenant le vecteur le plus court, et le "second plus court" : le vecteur le plus court indépendant du premier (cela ne se généralise pas en dimension 4 et au delà). La méthode de Gauss la calcule ainsi. Soit (U, V) la base initiale, avec V le vecteur le plus long. Soit W la projection de V sur U. Soit $kU, k \in \mathbb{Z}$ le vecteur le plus près de W. Si $k = 0$, alors la base ne peut être réduite et est minimale. Sinon $W - kU$ est plus court que V et toujours indépendant de U. Recommencer sur la base $U, W - kU)$.

Calcul d'inverse modulo P Théorème de Fermat : si P est premier, alors $x^{P-1} = 1 \bmod P$, pour tout x non nul. Donc l'inverse de x modulo P est $x^{-1} = x^{P-2} \bmod P$. Utiliser l'exponentiation rapide, qui nécessite $O(\log P)$ opérations, sur des nombres entiers inférieurs à P^2

Autre méthode possible,de même complexité : utiliser l'algorithme d'Euclide étendu. Pour (a, b) donnés, il calcule (u, v) tel que $au + bv = \text{pgcd}(a, b)$. L'appliquer pour $(a, b) = (x, P)$. On obtient (u, v) tels que $xu + Pv = 1$ donc $x^{-1} = u$ modulo P.

Critère d'Euler a est un carré modulo P (P premier) ssi $a^{(P-1)/2} = 1 \mod P$. Preuve : s'il existe x tel que $x^2 = a \mod P$, alors $x^{P-1} = 1$ par Fermat ; or $x^{P-1} = a^{(P-1)/2}$.

Racine carrée dans $Z/P\mathbb{Z}$ Si $P = 3 \mod 4$ et que a est un carré, alors sa racine carrée est $a^{(P+1)/4}$ modulo P. Si $P = 1 \mod 4$, on utilise la méthode de Zassenhauss-Cantor (expliquée dans le livre de Naudin et Quitté : Algorithmique géométrique, empruntable à la bibliothèque en 51 8.4–1099). Voici un exemple. $P = 13$ et on veut α tel que $\alpha^2 = a = 10$. Pour tous les nombres $t + \alpha$ dans $\mathbb{Z}/P\mathbb{Z}$, on doit avoir, d'après fermat : $(t + \alpha)^{(P-1)/2} = \pm 1$. La méthode choisit des t au hasard dans $Z/P\mathbb{Z}$ jusqu'à en trouver un qui convienne, et calcule "formellement" $(t + \alpha)^{(P-1)/2} = u + v\alpha$: ils sont considérés comme des polynômes en α, dont les coefficients sont dans $\mathbb{Z}/P\mathbb{Z}$, et en tenant compte de : $\alpha^2 = a = 10$. On obtient dans cet exemple, selon les valeurs de t :

$t = 1 \Rightarrow (1 + \alpha)^6 = 12 + 0\alpha = -1$

$t = 2 \Rightarrow (2 + \alpha)^6 = 0 + 2\alpha = \pm 1$

$t = 3 \Rightarrow (3 + \alpha)^6 = 1 + 0\alpha = 1$

$t = 4 \Rightarrow (4 + \alpha)^6 = 0 + 11\alpha = \pm 1$

$t = 5 \Rightarrow (5 + \alpha)^6 = 0 + 2\alpha = \pm 1$

$t = 6 \Rightarrow (6 + \alpha)^6 = 7 + 12\alpha = 1$ ou 0 selon que $\alpha = 6$ ou 7.

$t = 7 \Rightarrow (7 + \alpha)^6 = 7 + 1\alpha = 1$ ou 0 selon que $\alpha = 7$ ou 6.

$t = 8 \Rightarrow (8 + \alpha)^6 = 0 + 11\alpha = \pm 1$

$t = 9 \Rightarrow (9 + \alpha)^6 = 0 + 2\alpha = \pm 1$

$t = 10 \Rightarrow (10 + \alpha)^6 = 1 + 0\alpha = 1$

$t = 11 \Rightarrow (11 + \alpha)^6 = 0 + 11\alpha = \pm 1$

$t = 12 \Rightarrow (11 + \alpha)^6 = 12 + 0\alpha = -1$

Il y a 3 cas. Le premier cas $u = 0, v \neq 0$ survient pour beaucoup de valeurs de t, ici $2, 4, 5, 8, 9, 11$; alors $v\alpha = 1$ (ou -1), donc $\alpha = \pm v^{-1} \mod P$. Dans le deuxième cas, pour t dans 1, 3, 10, 12, il y a échec : $v = 0, u = \pm 1$ et on ne peut calculer α. Enfin dans le troisième cas, $t = 6, 7$ égale α, mais ce cas ne se produit que 2 fois sur $P - 1$. En moyenne, 2 ou 3 essais de t suffisent pour se

trouver dans le premier cas. Voir pg 316-321 du livre de Naudin-Quitté de 1992.

Référence Voir le livre de Ian Stewart : L'univers des nombres, chez Belin, Pour la Science, empruntable à la bibliothèque en 513–1006, pages 101–105, chapitre : Le théorème de Noël.

42 Arbres équilibrés. 1 personne. Ocaml.

Programmer en Ocaml une librairie de gestion des arbres équilibrés (chap. de IA). Vous utiliserez des foncteurs, et le type des éléments et la fonction de comparaison sera passée en paramètre.

Vous testerez avec la méthode de Dijkstra. Vous comparerez avec un arbre naïf (non équilibré).

43 Générateur aléatoire. Ocaml. 1 ou 2 étudiants.

Encadrant : éventuellement, O. Bailleux.

Trouver dans la littérature des méthodes de génération de suites aléatoires d'entiers (pseudo aléatoires serait plus juste, ces méthodes étant déterministes), ainsi que des méthodes pour tester la qualité statistique de ce pseudo hasard, et l'imprédictibilité des suites générées. Programmer les, et tester les. Attention : ce n'est pas si facile que vous le croyez.

Soignez votre rapport : il doit pouvoir être lu comme un cours d'introduction à ce problème.

Si possible, soumettez les suites que vous avez engendrées au programme de Sloane.

Applications : cryptographie, sécurité, confidentialité.

44 Génération d'anagrammes plausibles de patronymes. Ocaml. 1 étudiant

A partir d'un prénom P et d'un nom N (en fait des lettres présentes dans le prénom et le nom : L), vous générerez un anagramme plausible : un autre prénom P' et un autre nom N'. Vous utiliserez un dictionnaire de prénoms. La principale difficulté est de générer un nom plausible avec les lettres restantes (non utilisées dans le prénom). Jean Véronis propose un tel logiciel sur son blog.

Pour générer un nom plausible, vous pouvez calculer, à partir de la liste de tous les prénoms, une chaîne de Markov : M_{ij} est la probabilité que la lettre i soit suivie de la lettre j. Vous pouvez ajouter 2 lettres virtuelles, pour le début et la fin du nom. Vous choisissez ensuite la permutation des lettres restantes pour le nom qui a la probabilité la plus grande.

Le programme de Jean VERONIS génère, à partir de : "jean veronis", les anagrammes : "Jenna VOIRES", "Jonis AVENER, "Joane RIVENS", "Javier NONSE", "Joann VEISER", etc.

45 PSLQ. 2 étudiants. Ocaml

L'algorithme PSLQ permet de découvrir des relations entières entre des nombres réels décrits par des approximations : soit $x_1, \ldots x_n$ n nombres réels, approchés par des flottants ou des rationnels ; une relation entière est un ensemble d'entiers a_i tels que $\sum a_i x_i$ est un petit nombre en valeur absolue (et les a_i ne sont pas tous nuls !). Un exemple trivial est : $x_1 = 0.6666667$ et $x_2 = 1$; alors $a_1 = 3$, $a_2 = -2$ est une solution, qui permet de voir que "vraisemblablement", $x_1 = 2/3$. Autre exemple : $x_1 = 1.6180339887498949$, $x_2 = x_1^2 = 2.6180339887498949$, $x_3 = 1.$; alors la relation entière $a1, a2, a3 = -1, 1, -1$ suggère que x_1 est en fait la racine de $x^2 - x - 1 = 0$, autrement dit $x_1 = \frac{1+\sqrt{5}}{2}$. Pour fabriquer des exemples, vous résoudrez des équations algébriques à coefficients entiers, et vous chercherez des relations entières entre les puissances des racines ; vous pourrez ainsi vérifier que vous retrouvez bien les équations algébriques initiales. Parfois, si le polynôme est factorisable, vous trouverez des relations plus simples (des équations algébriques de degré plus bas) : c'est une façon de factoriser les polynômes. Vous tenterez de généraliser votre programme aux nombres complexes. Il y a beaucoup de documents sur la toile à propos de la méthode PSLQ et ses variantes (par exemple "A gentle introduction to PSLQ" de Armin Straub, et "PSLQ : an algorithm to discover integer relations" par David Bailey et J.M. Borwein).

46 Ensembles de Julia et intervalles. Ocaml. 2 étudiants

Pour $c \in \mathbb{C}$, l'ensemble de Julia J_c est l'ensemble des points z du plan complexe tels que l'orbite de z : $\{z, f(z), f(f(z)), f(f(f(z))), \ldots\}$, où $f(z) = z^2 + c$, ne va pas à l'infini. Vous afficherez ces ensembles avec la méthode décrite par Afonso Paiva, Jorge Stolfi et Luis Henrique de Figueredo dans l'article "Robust visualization of strange attractors using affine arithmetic" Computers & Graphics 30 (6), 2006 pages 1020–1026 [**?**]. Vous n'êtes pas obligé d'utiliser une arithmétique affine, une arithmétique d'intervalles est suffisante. Le calcul

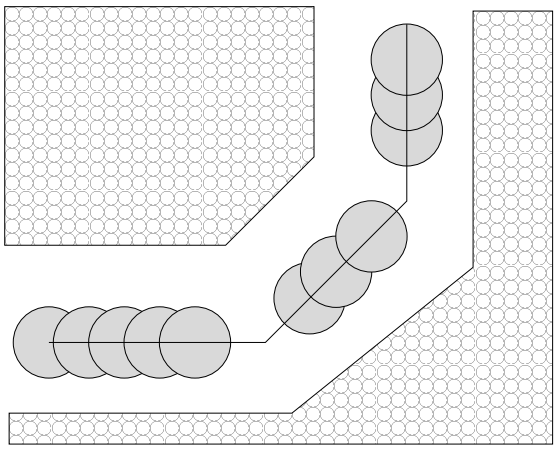
des composantes fortement connexes d'un graphe orienté est expliqué dans le livre "Introduction à l'algorithmique" de Cormen, Leiserson, Rivest et Stein [**?**]. Des valeurs de c donnant de "beaux" ensembles de Julia sont mentionnés dans wikipedia. Voir le fichier figueiredo.pdf dans le répertoire UTILITES.

47 SVD, GSVD, et compression d'images de visages. Ocaml. 2 étudiants

Vous programmerez la méthode présentée par Hervé Abdi dans "Singular Value Decomposition (SVD) and Generalized Singular Value Decomposition (GSVD)" pour compresser des fichiers d'images de visage. (voir fichier SVD.pdf dans le répertoire UTILITES). Vous pouvez utiliser la librairie Ocamlgsl, qui est une interface avec la GNU Scientific Library. Demandez le fichier du trombinoscope à DM.

48 Planification de trajectoire. 1 ou 2 étudiants. Ocaml

Un robot est décrit par un ensemble de tiges articulées, chaque tige est décrite par un ensemble de disques. Les obstacles sont décrits, eux aussi, par un ensemble de disques. Trouver une trajectoire pour le robot pour aller d'une configuration donnée à une autre. Une configuration est donnée par la position de la "tête" du robot (un disque particulier) et les angles entre les tiges. Une configuration est licite ssi le robot ne touche pas les obstacles et si les tiges ne se heurtent pas entre elles. Méthode : échantillonner l'espace des configurations, avec une résolution de plus en plus fine. Dans les premières étapes, de petits heurts sont tolérables (par exemple en réduisant le rayon des disques du robot) ; quand une trajectoire grossière a été trouvée, elle est améliorée en échantillonnant davantage l'espace des configurations près de cette trajectoire. Vous utiliserez la notion d'écart (ou écartement) : l'écart d'un point M par rapport au plus court chemin de A à B est $AM + MB - AB$ (où AM, MB, AB sont trois plus courts chemins). L'écart d'un sommet permet de savoir s'il faut échantillonner le disque centré en ce sommet. Jean-Paul Laumond a écrit de nombreux articles sur ce type de méthodes.



49 Faire un créneau, et autres manoeuvres en voiture. Ocaml. 1 ou 2 étudiants.

La librairie graphics.cma est suffisante.

On considère le squelette d'une voiture (ou d'un tricycle, pour commencer), en 2D. L'état de la voiture est décrit par l'angle des roues avant (0 si la voiture va tout droit), la position et l'orientation (un angle) d'un point de la voiture (disons le milieu des 2 roues avant). Vous vous donnez deux états différents de la voiture. De plus il y a un espace libre, que peut utiliser la voiture. Il vous faut générer une trajectoire "visuellement réaliste" de la voiture entre les 2 états (par exemple un créneau), et de longueur raisonnablement courte. Pour le principe d'un algorithme, lire le sujet : Planification de trajectoire. Attention ! les deux sujets sont quand même différents. Avec celui ci, la configuration (l'angle du volant) définit le mouvement.

50 Problème du voyageur de commerce et recuit simulé. Ocaml. 1 étudiant.

La librairie graphics.cma est suffisante. Vous résoudrez le problème du voyageur de commerce ("Travelling salesman problem") avec l'heuristique du recuit simulé ("simulated annealing"). Les villes seront données par leurs coordonnées 2D, et la distance entre deux villes est la distance euclidienne. Une des méthodes part d'une permutation aléatoire et remplace itérativement deux segments AB et CD par les deux segments AD et BC, ou par AC et BD, selon le plus court. Vous trouverez de nombreux exposés de la méthode sur internet, par exemple wikipedia.

51 Problème du voyageur de commerce. Ocaml. 1 étudiant.

La librairie graphics.cma est suffisante. Vous résoudrez le problème du voyageur de commerce ("Travelling salesman problem") avec l'heuristique de votre choix (algorithme génétique, colonies de fourmis ("ant colony optimization")), etc.

52 Triangulation d'un ensemble de points 2D. Ocaml. 2 étudiants.

En modifiant une méthode de calcul d'enveloppe convexe, vous triangulerez un ensemble donné de points 2D. Puis, par échange d'arêtes, vous calculerez la triangulation de Delaunay. Lire : "An incremental algorithm based on edge swapping for constructing restricted Delaunay triangulations", par Marc Vigo Anglada, disponible sur internet, ou bien le polycopié de Franck Hétroy : Un petit peu de géométrie algorithmique `http://evasion.imag.fr/Membres/Franck.Hetroy/Teaching/GeoAlgo/poly_geoalgo.pdf`. Utilisez des coordonnées entières pour éviter les difficultés dues à l'imprécision numérique.

53 Le jeu du wumpus. Ocaml. 4 étudiants

Empruntez le Livre : "Intelligence Artificielle", par Stuart Russell et Peter Norvig. Chapitre 7. Disponible BU cote 518.1.1070.

Il s'agit de simuler le comportement d'un agent dans un monde virtuel. Ce monde est un ensemble de salles (les carrés d'une grille) dont l'agent connait l'agencement. L'agent sait qu'il y a un monstre dans l'une des salles, le wumpus. Le wumpus peut être détecté à cause de son odeur, dans les cases voisines. L'agent perçoit les odeurs de la salle où il se trouve, et il a une mémoire. L'agent doit aussi éviter les salles avec des puits sans fond, perceptibles par une brise dans les salles voisines. L'agent perçoit la brise dans la salle où il se trouve, et il a une mémoire. L'agent dispose d'un arc et d'une seule fléche pour tuer le wumpus. L'agent doit tenter de trouver un trésor dans une des salles. Les connaissances de l'agent sont représentées par de la logique des propositions et des variables logiques (wumpus-24 : le wumpus est dans la salle 2, 4 ; brise-1-2 : il y a une brise dans la salle 1, 2 ; odeur-2-3 : il y a une odeur dans la salle 2, 3 ; agent-1-2 : l'agent se trouve dans la case 1, 2 ; etc). Voici un exemple de régles : wumpus-24 =¿ odeur-14. En fait, les variables sont indicées par une date entiére. Par exemple, agent-1-2-0 signifie que l'agent est en 1, 2 au temps 0. Le raisonnement de l'agent et la planification de ses actions sont réduits 'a des problémes de satisfaction de contraintes booléennes, le probléme SAT. Vous

écrirez votre propre solveur SAT, pour des univers de 4 par 4 cases. Vous pouvez aussi utiliser des algorithmes évolutionnaires pour planifier les actions de l'agent, et sélectionner les meilleures stratégies.

Vous introduirez des variantes pour vous démarquer de la solution proposée par la REF. Variante possible : vous pouvez aussi simuler le comportement du wumpus, ou de plusieurs agents. Vous visualiserez le monde du wumpus, par exemple avec des icônes pour représenter le wumpus et l'agent.

Les méthodes de représentation des connaissances et de planification utilisées dans cette application ludique sont aussi utilisées dans des applications "sérieuses", commerciales (ex : robot sur internet, cherchant les machines à laver les moins chéres ; ceci est aussi appelé : l'"ingéniérie ontologique"), industrielles (job scheduling, logistique, "model checking"), militaires.

54 Pavages non périodiques. 1 étudiant

La librairie graphics de Ocaml est suffisante. Empruntez le livre [**?**] "Géométrie des pavages" de P. Audibert à la BU (cote 516/1181) et programmez la méthode multigrille (figures pg 236–237 du livre).

55 Pavages non périodiques. 1 étudiant

La librairie graphics de Ocaml est suffisante. Empruntez le livre [**?**] "Géométrie des pavages" de P. Audibert à la BU (cote 516/1181) et programmez le pavage quasi périodique presque orthogonal, pg 218 du livre.

56 Pavages non périodiques. 1 étudiant

La librairie graphics de Ocaml est suffisante. Empruntez le livre [**?**] "Géométrie des pavages" de P. Audibert à la BU (cote 516/1181) et programmez le pavage des Fig. 7.9, 7.10, 7.12, 7.13 du livre.

57 Contraintes sur des courbes de subdivision. 3 étudiants

Le but est la résolution de contraintes géométriques sur des courbes 2D subdivisées par la méthode de Chaikin.

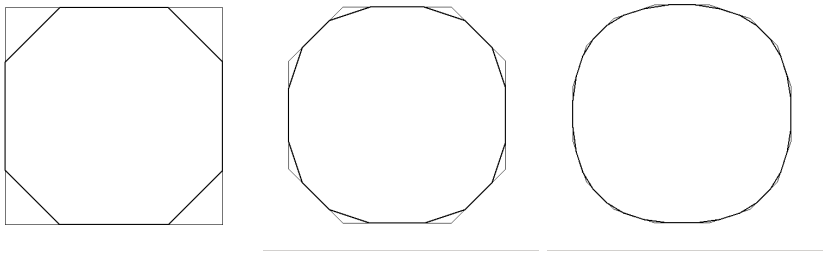


FIGURE 1 – Trois étapes de la subdivision de Chaikin.

La subdivision de Chaikin (Figure 1) coupe au quart et au trois quart chaque segment d'un polygone $P = (P_i, i = 0...n-1)$; le polygone résultant est $Q = (Q_i, i = 0..2n-1)$, où $Q_{2k} = (3P_k + P_{k+1})/4$ et $Q_{2k+1} = (P_k + 3P_{k+1})/4$. Chaque étape de subdivision double donc le nombre de sommets et d'arêtes du polygone. Le polygone initial P génère une suite de polygones $Q, R, S, T, U, \ldots$ qui après 5 ou 6 étapes est visuellement indiscernable d'une courbe lisse. Certains sommets du polygone initial peuvent être marqués comme indestructibles, et persisteront dans les subdivisions ultérieures.

Vous chercherez quels sommets du polygone initial font que la courbe limite, ou le polygone après un nombre fixé de subdivisions, passe par des points donnés. Vous vous donnerez un polygone initial, et vous vous donnerez des points de passage proches de la coube limite du polygone initial pour avoir des chances de réussir.

Pour résoudre les contraintes, vous n'utiliserez pas une méthode qui exige de poser explicitement un système d'équations. Il suffit en effet de pouvoir mesurer à quel point un polygone donné, obtenu après quelques étapes de subdivision, satisfait les contraintes spécifiées. Il est facile de mesurer la distance entre le polygone et un point par lequel il est censé passer. Pour résoudre, vous minimiserez la somme des distances entre la courbe et les points de passage. Vous testerez la méthode de descente de gradient (il faudra donc évaluer le vecteur gradient, numériquement), le simplexe de Nelson-Mead ou de Torczon, qui sont présentées ici :

`http://transp-or.epfl.ch/courses/optimization2011/slides/11b-sans-derivee.pdf`

(voir aussi dans le répertoire UTILITIES). Eventuellement l'optimisation par essaim ("swarm optimization") dite aussi optimisation particulaire (PSO sur google) vous donnera de bonnes approximations de solution, que vous améliorerez par une des méthodes précédentes.

La librairie graphics.cma est suffisante. Une jolie interface IHM n'est pas demandée.

Remarque. D'autres contraintes sont possibles : tangence à une droite donnée, ou à un autre polygone de Chaikin, ou à une courbe donnée par une équation paramétrique (exemple : $x = \cos t, y = \sin t$) ou une équation implicite ($x^2 + y^2 - 1 = 0$).

Références

[ABFM08] Carlos Ansótegui, Ramón Béjar, César Fernàndez, and Carles Mateu. Edge matching puzzles as hard sat/csp benchmarks. In *CP '08 : Proceedings of the 14th international conference on Principles and Practice of Constraint Programming*, pages 560–565, Berlin, Heidelberg, 2008. Springer-Verlag. http ://www.springerlink.com/content/m574272363128136/.

[Aud13] Pierre Audibert. *Géométrie des pavages*. Lavoisier, 2013. cote BU Dijon : 516/1181.

[CJB07] A. Couturier and G. Jean-Baptiste. *Programmation fonctionnelle appliquée aux calculs scientifiques : Objective CAML. Méthodes numériques & applications*. Number vol. 1 in Les Cours de l'ICES. Cépaduès éd., 2007.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[CMP98] Emmanuel Chailloux, Pascal Manoury, and Bruno Pagano. *Développement d'applications avec Objective Caml*. O'Reilly France, 1998. http ://caml.inria.fr/pub/docs/oreilly-book/.

[CMP00] Emmanuel Chailloux, Pascal Manoury, and Bruno Pagano. *Developping Applications with OCAML*. Oreilly book, 2000. http ://caml.inria.fr/pub/docs/oreilly-book/.

[Har07] Jon D. Harrop. *OCaml for Scientists*. May 2007. http ://www.ffconsultancy.com/products/ocaml_for_scientists/.

[Heu08] Marijn J.H. Heule. Solving edge-matching problems with satisfiability solvers. pages 88–102. University of Leuven, 2008. http ://www.st.ewi.tudelft.nl/~marijn/publications/eternity.pdf.

[Hic08] Jason Hickey. *Introduction to Objective Caml*. 2008.

[PdFS06] Afonso Paiva, Luiz Henrique de Figueiredo, and Jorge Stolfic. Robust visualization of strange attractors using affine arithmetic. *Computers & Graphics*, 30(6) :1020–1026, december 2006. http ://www.tecgraf.puc-rio.br/ lhf/ftp/doc/sa.pdf.

[PS98] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization : algorithms and complexity*. Dover, 1998.

[PTVF92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C, the Art of Scientific Computing*. Cambridge University Press, 1992.