

Projets algorithmique et complexité

M1 informatique, 2023

2023

Contents

1	Introduction	2
2	Pourquoi Ocaml est il imposé ?	3
3	La notation du projet	3
4	Eternity II. 1 étudiant	3
5	Eternity II et SAT. 2 étudiants	4
6	Pavage en carrés. 2 étudiants	4
7	Solitaire. 1 étudiant	5
8	Calcul de π . 1 étudiant	5
9	Taquin. 2 étudiants	5
10	Cube de Rubik. 2 étudiants	5
11	Percolation. 1 étudiant	5
12	Multiplication de Strassen. 1 étudiant	6
13	Interpolation. 1 ou 2 étudiants	6
14	Méthode de Dijkstra. 1 étudiant	6
15	Stéganographie. 2 étudiants	6
16	Coloration de graphes. 1 étudiant	7
17	Othello et optimisation. 2 ou 3 étudiants	7
18	Arbres équilibrés. 1 étudiant	7
19	Génération d'anagrammes plausibles de patronymes. 1 étudiant	8

20 Ensembles de Julia et intervalles. 2 étudiants	8
21 SVD, GSVD, et compression d'images de visages. 2 étudiants	8
22 Planification de trajectoire. 2 étudiants.	8
23 Problème du voyageur de commerce et recuit simulé. 1 étudiant	9
24 Triangulation d'un ensemble de points 2D. 2 étudiants	9
25 Références	10

1 Introduction

Le rapport écrit doit être réalisé en Latex (via des outils comme texlive, MikTeX ou www.overleaf.com). Ce texte est rédigé en Latex. Le fichier de style latex et le fichier bib (pour la bibliographie) sont disponibles sur plubel, ainsi qu'un fichier d'exemple.

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.

Il doit comprendre :

1. une présentation du projet
2. une explication du ou des algorithmes utilisés (algorithme et pseudo-code devront être présents)
3. une analyse rapide de la complexité (si elle est fait par ailleurs, vous pouvez vous y référer)
4. les détails pertinents de votre implémentation (points durs,
5. des résultats et leurs analyses (exactitude, stabilité, régularité, évolution itérations/temps...). Vous devez afficher les courbes de temps/itérations, une sortie de votre programme OCaml en fichier gnuplot ou csv est pratique pour l'insertion à la volée dans Latex.
6. une conclusion qui établit (ou non) en justifiant le lien entre votre implémentation et l'analyse théorique de l'algorithme.

Rédigez votre rapport avec soin. La langue naturelle est le premier langage de programmation et de modélisation, 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` peut vérifier l'orthographe, comme LanguageTool (<https://languagetool.org/fr>).

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 `ocaml-doc` (similaire à `javadoc`).

Vous créez une archive de votre projet au format ZIP, 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).

L'archive doit contenir les fichiers supports de votre présentation orale. Par contre, prévoyez un fichier `LISEZMOI.md` (au format Markdown), un `makefile`, et une commande `"make clean"`, `"make all"`, et éventuellement quelques programmes de démonstration, que vous appellerez `demo1`, `demo2`, etc.

2 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 pouvez utiliser des "références" pour programmer avec des affectations.
- 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).

Attention : vous n'avez aucune contrainte sur la sortie graphique de votre programme, ce n'est pas l'objet de ces projets. La bibliothèque `Graphics.cma` peut être utilisée mais on privilégiera une sortie texte intégrable dans LaTeX, comme `gnuplot`, `csv` (l'import et la mise en forme de tableau sont possible facilement : <https://bioinfo-fr.net/latex-automatisez-traitemnt-csv>), `tex` ou `pstTricks`.

3 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 sera de 0 pour le ou les groupes.

4 Eternity II. 1 étudiant

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.

5 Eternity II et SAT. 2 étudiants

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 ([Heu08, ABFM08] par exemple).

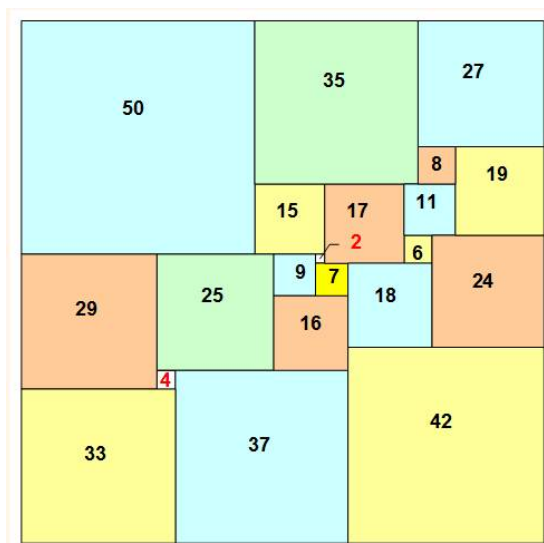
Vous générerez des puzzles aléatoires du jeu Eternity II. Vous générerez le problème de contraintes booléennes correspondant, que vous sauverez dans un fichier ; vous appellerez le solveur de contraintes booléennes ; vous lirez le fichier solution. Vous testerez quelles tailles de puzzle sont solubles 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.

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.

6 Pavage en carrés. 2 étudiants

Soit un carré de côté 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 :



7 Solitaire. 1 étudiant

Un carré de 2×2 cases est enlevé aux quatre coins d'un damier 7×7 . Il reste donc $7^2 - 4 \times 4 = 33$ 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 essaieriez ensuite d'agrandir le damier ; jusqu'où pouvez vous aller ?

8 Calcul de π . 1 étudiant

Empruntez le livre de Jörg Arndt et Christoph Haenel : "A la poursuite de Pi", éditions Vuibert, à la BU (cote 510/1232). Programmez **au moins 3** algorithmes parmi les 9 du livre. Peut-être y a t'il encore le livre Jean-Paul Delahaye à la BU : "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 !

9 Taquin. 2 étudiants

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 ?

Liens : outre wikipedia, <http://maths.amateurs.fr/index.php?page=taquin>

10 Cube de Rubik. 2 étudiants

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

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>

11 Percolation. 1 étudiant

Une grille 2D est partitionnée en nn 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.

12 Multiplication de Strassen. 1 étudiant

La méthode de Strassen permet de multiplier des matrices carrées n par n en moins de temps que $O(n^3)$. Vous programmerez cette multiplication, ainsi que l'inversion basée sur la multiplication rapide de Strassen [CLRS01]. 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 M , $M - 1$ 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.

13 Interpolation. 1 ou 2 étudiants

Vous calculerez la courbe algébrique $f(x, y) = \sum_i \sum_j a_{ij} x^i y^j = 0$, avec $i + j \leq d$ de degré d donné qui passe, ou approche, un ensemble donné de points dans le plan. Vous utiliserez une résolution aux moindres carrés [CLRS01, PTVF92].

Vous afficherez la courbe et les points (issus d'un fichier).

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.

14 Méthode de Dijkstra. 1 étudiant

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 [CLRS01], un tas binomial [CLRS01]. Vous pouvez récupérer sur internet des implantations des tas de Fibonacci et des tas binomiaux ; par contre, vous programmerez le tas (*heap*).

15 Stéganographie. 2 étudiants

Vous utiliserez le logiciel convert, ou xv, sous linux pour convertir les images dans des formats de fichiers facilement lisibles en Ocaml (PPM), en texte simple.

Vous pouvez aussi utiliser 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 originale de stéganographie consiste à sacrifier ces bits et à les remplacer par ceux d'un texte 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 [CLRS01].

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).

16 Coloration de graphes. 1 étudiant

Programmer et optimiser votre propre méthode de coloration de graphe. Utiliser un import texte dans OCaml pour pouvoir tester vos graphes.

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.

17 Othello et optimisation. 2 ou 3 étudiants

Vous utiliserez le *minimax* avec élagage alpha-beta fourni dans [CMP00, CMP98], disponible librement sur internet (ou à la BU).

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 devez lire quelques articles sur les algorithmes évolutionnaires, inspirés de la théorie Darwinienne de l'évolution (swarm optimization, genetic algorithms, metaheuristics).

18 Arbres équilibrés. 1 étudiant

Programmer en Ocaml une librairie de gestion des arbres équilibrés. 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).

19 Génération d'anagrammes plausibles de patronymes. 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.

20 Ensembles de Julia et intervalles. 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 : $z, f(z), f(f(z)), f(f(f(z))), \dots$, 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 [PdFS06].

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 [CLRS01]. Des valeurs de c donnant de "beaux" ensembles de Julia sont mentionnés dans wikipedia.

21 SVD, GSVD, et compression d'images de visages. 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. Vous pouvez utiliser la librairie Ocamlgsl, qui est une interface avec la GNU Scientific Library.

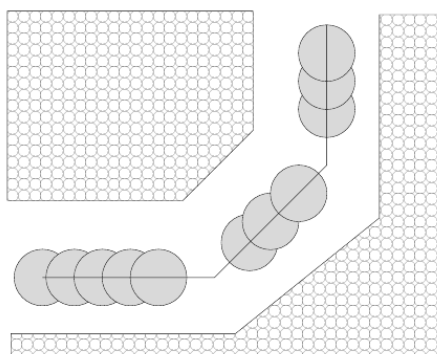
22 Planification de trajectoire. 2 étudiants.

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.



23 Problème du voyageur de commerce et recuit simulé. 1 étudiant

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.

24 Triangulation d'un ensemble de points 2D. 2 étudiants

En modifiant une méthode de calcul d'enveloppe convexe, vous triangulerez un ensemble donné de points 2D (issus d'un fichier texte). 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.

25 Références

ABFM08] Carlos Ansótegui, Ramón Béjar, César Fernández, 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. Developing 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.