D'un détail d'implémentation vers un nouveau regard sur l'interpénétration entre la théorie et la programmation

– pour rompre un enchantement secret de masse –

Daniel Porumbel, CEDRIC-CNAM

> J'ai suivi paisiblement le cours de mes méditations ; il est vrai de dire que j'y étais comme contraint et entraîné par une sorte d'instinct irrésistible. Mais cet instinct était fortifié d'une conviction réfléchie : j'estimais que la vérité qu'un homme a découverte, ou la lumière qu'il a projetée sur quelque point obscur, peut un jour [ma note : peu importe quand, même si très tard] frapper un autre être pensant, l'émouvoir, le réjouir et le consoler ; c'est à lui qu'on parle, comme nous ont parlé d'autres esprits semblables à nous et qui nous ont consolés nous-mêmes dans ce désert de la vie.
>
> Arthur Schopenhauer

Cette communication met un coup de projecteur sur les difficultés qui apparaissent sur le point de contact entre la théorie et la programmation. C'est sur ce point de contact que la courroie de transmission (de la force intellectuelle) perd une très grande partie de son énergie lors de l'implémentation pratique d'un algorithme mathématique. La traduction d'un langage de haut niveau en langage machine est moins critique, car les compilateurs modernes peuvent automatiser cela assez efficacement. Le plus difficile est de traverser tout les jours la frontière qui sépare le pays de la programmation du pays de la théorie ; la vie d'un travailleur frontalier n'est pas toujours si facile qu'on le pense. Comment expliquer cela ? Les conditions de vie dans ces deux pays exigent des traits intellectuels assez différents et parfois des vocations scientifiques presque opposées ; et en plus, il faut s'adapter à chaque fois qu'on traverse la frontière, c.-à.d., il faut passer d'un mode de travail à un autre.

Parmi les gens les plus attachés à la théorie il y a beaucoup qui ont une compréhension très approximative de la programmation et du génie logiciel. Cela est surtout vrai s'ils pensent que la programmation « c'est de la cuisine ; ça fait partie des petites choses qui doivent être faites pour obtenir des résultats ». Tant que cette vision reste au pouvoir, nous sommes sous l'emprise d'une sorte d'ensorcèlement systémique qui fracture la communauté en deux :

(a) D'un coté, des sommités académiques qui n'ont jamais vu un algorithme en exécution qui peuvent manquer de profondeur dans leur réflexions. Par exemple, leur discours se focalisent souvent sur les « grandes idées », sans réaliser que la dynamique et l'efficacité d'un algorithme peut facilement changer si on modifie un « petit » paramètre ou même si on introduit une contrainte redondante dans le modèle. Et il est impossible de pénétrer au cœur d'une telle question si on l'étudie à partir des hauteurs d'une théorie (simplificatrice). Cela les conduit à toute sorte d'erreurs d'approximation dans leurs jugements et dans leurs évaluations. Qu'on ne s'illusionne pas : quiconque n'a jamais lutté avec une machine pour contrôler la dynamique d'un algorithme ne connaitra jamais les secrets de l'algorithmique au delà de certaines généralités simplificatrices.

(b) D'un autre coté, les programmeurs qui voient tous les jours des algorithmes en exécution n'ont pas toujours – surtout les très très jeunes – l'esprit de synthèse, les grilles d'analyse et les outils cognitifs les plus appropriés pour déconstruire certains pièges du système (et Dieu sait qu'il y en a). Cela les empêche de développer une vision vraiment claire et complète sur la place et la valeur de leur travail dans un contexte (scientifique) plus général.

Ces idées non-techniques se matérialisent et se projettent sur le plan technique de milles façons. On doit choisir entre (a) ou (b) plus souvent qu'on ne le pense.

Combien d'étudiants se trouvent au début de leur thèse désarmés quant à la meilleur manière de coder les algorithmes ou les modèles du directeur de thèse ? Un autre volet à étudier c'est l'offre

de formations proposée habituellement en RO et en informatique. On trouve souvent beaucoup de cours sur des techniques de programmation, compilation et génie logiciel. On trouve aussi beaucoup de cours sur diverses théories (de RO). Mais, à mon avis, il n'y a pas assez de cours qui essayent de « marier » les deux. Je ne parle pas de simple TPs de mise en application d'une théorie, cela est assez repandu ; je parle de cours avec un intitulé qui montre qu'il s'agit en premier lieu d'un cours sur l'étude des points de contact entre théorie et programmation.

Voici un slide extrait d'un exposé ISMP 2018 (de A.V. Goldberg) :

```
What students learn?
* Theory of Algorithms
    - Ignore constant factors for machine independence
    - Analysis mostly worst-case
* Programming Languages
    - Aim for compiler to take care of all low-level details
    - High-level specification
* Software Engineering
    - Correctness
    - Development Speed
    - Portability, maintainability and testability
RED: Tempting to ignore constant factors and low-level details
```

Ce slide concerne plutôt les États Unis. En France, on peut ajouter à cette liste les cours master en RO sur des modélisations linéaires, décompositions, inégalités valides, facettes, (méta)-heuristiques, etc. Mon impression c'est que beaucoup (trop) de gens focalisent souvent leur enseignements sur les « grandes idées », en déconnectant ces idées des aspects programmation qui sont presque oubliés. Mais peut-on prendre les meilleures décisions algorithmiques dans la vie réelle de tous les jours en ne prenant en compte que les « grandes idées » ? Pour ma part, je ne le crois pas du tout. En tout cas, on trouve rarement une formation qui essaie de lier les volets "Theory of algorithms" et "Programming languages" du slide ci-dessus. Donc on oublie un peu d'apprendre aux étudiants l'art de coder un algorithme théorique déjà conçu le plus efficacement possible. Une fois qu'on a prouvé toutes les inégalités valides et qu'on a bien décomposé le modèle, que doit-on faire ? Est-ce que le résultats observés lors de l'étape de programmation peuvent être fournis à leur tour à la théorie et ainsi construire un système à rétroaction (avec *feedback loop*) ? La difficulté de concevoir un petit cours sur cela confirme, à mon avis, le fait qu'une borne partie de l'énergie intellectuelle se perd dans la traduction de la théorie en pratique.

En ce qui concerne la recherche, j'ai rencontré une fois un chercheur en informatique qui se targue de proposer (à des conférences d'élite comme FOCS/SODA/STOC) des algorithmes conçus pour ne jamais être implémentés ! C'était un vrai artiste parce qu'il ne voulait pas savoir si ses travaux allaient un jour remplir une fonction (d'ordre pratique). L'esprit d'un programmeur ingénieur est le parfait contraire. Pour lui, la théorie seule n'a pas d'importance par elle même : c'est comme l'amour sans les œuvres. Son travail vise donc à donner vie à la théorie ; le but est toujours de mettre en mouvement un ordinateur, pour qu'il fasse toutes sortes de choses, aussi folles soient-elles.

Beaucoup pensent que la programmation doit rester une affaire secondaire tout simplement parce la théorie représente naturellement la très noble « recherche fondamentale ». Et je vous l'accorde : cela s'applique parfaitement aux 1% meilleurs chercheurs théoriques qu'on doit pas embêter avec des problèmes d'implémentation. Mais l'histoire n'est pas si rose ou si simple pour les autres 99% des chercheurs. Ils évitent souvent les questions d'implémentation pour des raisons très prosaïques : il faut prendre des instances, lancer et relancer des tests numériques encore et encore, se replonger dans à chaque révision du papier ; bref, c'est trop fastidieux. En conséquence, beaucoup se disent : je vais chercher une petite jolie théorie. Et si j'arrive à bien m'entendre avec les quelques dizaines de chercheurs qui travaillent sur cela, ça sera que du bonheur. »

On peut ainsi obtenir le droit d'appeler son travail « recherche fondamentale » sans qu'on puisse fournir des contre-arguments (trop) facilement. C'est assez complexe de combattre – ou plus précisément démasquer – certains gens qui abusent du langage lorsque ils disent travailler en « informatique fondamentale ». C'est vrai que la notion d'informatique fondamentale, qui peut exister de manière légitime, n'a pas toujours un impact direct et immédiat sur la vie pratique. Mais beaucoup ont mal

inversé cet argument (dans leur tête) pour tirer une conclusion illégitime : tout ce qui est loin de toute réalité terre-à-terre et sans impact pratique doit s'appeler « recherche fondamentale ». Voici un autre stratagème classique qui marche plus souvent qu'on ne le pense. [1] Ils choisissent une niche où ils se retranchent comme dans un refuge derrière une toile méticuleusement tisée qui peut contenir : des mots-clés de plus en plus audacieux, des expressions (très) pompeuses, des fioritures, des formules excessivement longues à perte de vue, le tout décrit d'une manière dialectiquement exagérée en utilisant le plus imperméable jargon. Cela peut (éboluir et) induire en erreur ceux qui pensent que tout ce qui est difficile à décrypter ou à comprendre doit contenir que des idées très savantes ou très fortes. Cette dense toile fait qu'il est difficile pour un extérieur d'y voir clair.

Un peu à l'image de l'art abstrait, les gens actifs dans une telle niche peuvent ainsi s'évaluer *uniquement entre eux*. Comparez cela à la peinture de la renaissance qui, sans être dépourvue d'un sens très profond, a produit une esthétique qui peut encore être appréciée par tout le monde. Il ne faut pas tomber dans le piège de penser que tout ce qui est difficile à déchiffrer doit contenir une vérité très profonde ; beaucoup de grandes vérités peuvent être exprimées par de très simples mots et par de simples formules. Il est peut-être plus sage d'éviter un écrivain ou un artiste qui ne s'est pas efforcé par tous les moyens d'être clair pour son lecteur – surtout si vous sentez qu'il veut épater en nous laissant croire qu'aucun mot ou formule n'est capable d'exprimer ses hautes et profondes pensées. La subtilité consiste à arranger le charabia de telle sorte que le lecteur doit penser qu'il est dans l'erreur s'il ne le comprend pas, alors que l'auteur sait parfaitement que c'est lui qui est en faute.

J'étais à deux doigts de tomber dans ce genre de piège. Ma liste de publications comporte un petit article sur un algorithme sans implémentation dans la théorie des fonctions sous-modulaires [1]. À l'époque, je me suis dit : « La théorie de ces fonctions est intéressante. On a pu gagner le prix Fulkerson (en 2003) pour des travaux là dedans. Tout est très noble dans ce genre de petit monde, un peu à l'abri des extérieurs. ». Ultérieurement, j'ai appris que le prix Abel 2021 a été en partie accordé (à László Lovász) pour des travaux sur ces fonctions sous-modulaires. De toute façon, cette voie serait un piège pour moi. Un jour j'ai compris que je suivais cette théorie pour chercher (inconsciemment) une planque, c.à.d., une niche (lisez : un refuge) parmi les innombrables théories que l'humanité a conçues. Comme je ne fait *pas* partie des 1% des gens les plus forts sur les aspects théoriques, j'ai réalisé que mes travaux purement théoriques ne pourront jamais se rapprocher du prix Fulkerson. Et c'est ainsi que j'ai aussi compris qu'une loi plus implacable risquait de gouverner ma vie : ceux qui choisissent une planque finissent au placard (de leur discipline au sens large).

Cela m'a poussé à éviter les niches complètement théoriques pour me placer sur un domaine qui touche plus de monde : la frontière entre la théorie et la programmation. Il ne faut pas comprendre que je veux fuir la théorie. J'ai écrit 100 pages sur la théorie de l'optimisation sémipositive définie ou SDP [2], une théorie plus complexe que la programmation linéaire (en nombres entiers). Mais je ne pense qu'à trouver un jour les moyens de la connecter à la programmation. Je ne cherche *pas*, par exemple, à l'utiliser pour concevoir des algorithmes théoriques d'approximation (genre Goemans-Williamson) ; c'est complètement exclu pour ma part.

Une grande et très subtile difficulté est la suivante : comment allier la théorie et la programmation, sans s'éloigner ni de l'une ni de l'autre ? Je pense que la meilleure manière de les garder ensemble c'est de les garder à l'intérieur d'un seul cerveau. Cette tache devient de plus en plus difficile avec l'age. Plus on est âgé, plus on fait le travail de programmation à contrecœur. Mais je trouve important de surmonter cet obstacle. *Un travail issu d'un cerveau composite, où une personne développe la théorie et une autre l'implémentation, ne peut avoir qu'une importance secondaire* dans mon cœur.

Concernant cette pratique, je parle à dessein comme si c'était une affaire de cœur, pour être clair que j'émets ici un avis purement personnel. Je pense que cette pratique qui coupe les liens entre la théorie et la programmation mène à une communauté *fracturée et refroidie* qui évolue comme un enfant dans

1. Le stratagème est classique car il est utilisé depuis plusieurs siècles au delà du monde des sciences exactes. Voici comme Schopenhauer l'a décrit dans son domaine, la philosophie : « Beaucoup s'abritent derrière un appareil imposant de longs mots composés, de phrases creuses embrouillées, de périodes a perte de vue, d'expressions nouvelles et inconnues, toutes choses dont le mélange donne un jargon d'aspect savant des plus difficiles a comprendre. [...] Ce truc consiste a écrire d'une façon obscure, c'est-a-dire incompréhensible ; la finesse est d'arranger son galimatias de manière a faire croire au lecteur que la faute en est a lui-même s'il ne le comprend pas ; tandis que l'écrivain sait très bien qu'il en est seul responsable, vu qu'il ne dit rien qui soit réellement compréhensible, c'est-a-dire clairement pensé. Sans cet artifice, Fichte et Schelling n'auraient pu mettre sur pied leur pseudo-gloire. »

un orphelinat : même le service technique le plus brillant ne peut pas compenser la démoralisation qui résulte du manque d'intérêt véritable pour le travail de programmation. Mais on peut faire abstraction de cet obstacle artificiel : *la notion d'alliage théorie–programmation à l'intérieur d'un seul cerveau trouvera un jour tout son sens*, un dignité supérieure à celle de toute niche théorique.

Pour quoi ? Parce que la théorie et la programmation sont si interpénétrées en RO (et informatique) que leur harmonie ne peut se réaliser que si on arrive à les faire former un couple solide qui réside dans un seul cerveau, sans *jamais* faire chambre à part. Il ne suffit pas de se donner rendez-vous une fois tous les 10-15 jours pour se raconter ce qu'on on a fait. Je suis arrivé à cette conclusion par expérience, et non pas par la lecture d'un livre. Les meilleurs décisions théoriques que j'ai pu prendre aux moments les plus essentiels ont toujours été le fruit d'une *connaissance directe des phénomènes algorithmiques à l'œuvre. Et pour comprendre très bien les phénomènes algorithmiques les plus fins, une compréhension théorique peut ne pas suffire ; on a davantage besoin d'une compréhension théorique éclairée et mise en mouvement par les innombrables constats offerts par la pratique et par la programmation. Si vous pensez que cela est une évidence, que tout le monde sait que le couple thèorie-programmation doit résider dans un seul cerveau, montrez moi des milliers de gens qui vivent ce crédo. Au contraire, la plupart des publications sont issues d'un cerveau composite : certains gens prennent le rôle de guide théorique et ils sous-traitent la programmation à d'autres.*

La racine de ce problème se trouve le fait qu'on risque de construir une communauté *fracturée et refroidie*, comme évoquée plus haut. Il est facile à trouver des super théoriques (retranchés) dans leurs niches. C'est pas trop dur non plus de trouver des super-développeurs. Et beaucoup pourraient faire les deux. Sauf que, je pense, ils ont choisi de ne pas tenter d'entreprendre ce double effort – souvent, parce qu'ils échoué à comprendre l'intérêt.

Je pense que la meilleure manière de maitriser à fond l'essence de tout algorithme à vocation pratique, c'est de l'implémenter soi même. Un exemple typique est l'algorithme de recherche locale. Il a l'air d'être facile à lire et à comprendre. Mais si on essaye de l'implémenter en utilisant que des connaissances théorique acquises rapidement, on aura des résultats lamentables sur un problème compétitif. La recherche locale peut paraître simple, mais il faut des mois de travail à la fois pratique et théorique pour être compétitif sur un problème bien étudie. Sinon, le potentiel d'un cerveau composite qui ignore les liens entre théorie et programmation risque de ne pas rester compétitif à long terme sur de tels problèmes. Le phénomène peut allez au delà des science dures. J'admets que cela dépend beaucoup du contexte, mais il y a des domaines où la valeur d'un cerveau composite peut être plus limitée que vous ne l'avais remarqué. La littérature est peut-être un bon exemple. Prenez votre livre préféré. Il a été écrit par une seule personne, sauf rare exception.

Un autre obstacle c'est qu'il reste très difficile de vendre ce concept de couple théorie–programmation qui réside dans un seul cerveau. Un tel combat et déjà perdu d'avance et déjà gagné d'avance. J'explique le paradoxe. C'est perdu d'avance parce que le système ne va jamais financer ou soutenir ce genre d'idées qui n'ont apparemment rien de grandiose. Le discours ici peut même donner l'impression de revenir à des éléments très simples et basiques, ou à un passé que beaucoup ont laissé « derrière eux ». Il est difficile de décrire cela en termes très modernes, ou même de glisser un petit mot à la mode, une expression avant-gardiste ou d'autres concepts pompeux « pour faire le buzz ». [2]

Mais à chaque fois qu'on décide de faire quelque chose qui n'est pas valorisé par le système, il y a toujours une petite chose gagnée d'avance. Le niveau de petitesse ou de grandeur n'est pas essentiel. Dans une telle démarche, on trouve toujours la satisfaction de prendre partie *d'un combat d'un homme libre contre l'homme aspiré par le système et par la collectivité*. Au moins, tu ne grandit pas comme « another brick in the wall ». Et cela donne aussi une autre satisfaction qu'on oublie trop souvent : c'est la rareté qui fait le prix de la marchandise. Les hommes aspirés par la collectivité sont si nombreux qu'ils peuvent se perdre et se faire dissoudre dans la masse ; s'ils n'arrivent jamais à se démarquer de la foule, ils seront oubliés en fin de compte. Cela sera encore plus vrai le jour quand notre système (scientifique) va commencer à céder sous le poids d'une masse de plus en plus grande d'hommes qui pensent (infiniment) plus aux récompenses de l'adhésion au système qu'à l'examen de la vérité. Si jamais cela se produit à trop grande échelle, qui va continuer à les prendre tous au sérieux ?

Ces idées n'ont pas pour but de suggérer des conseils (à tout-va pour tout le monde) ; je ne donne que des avis, j'expose et j'impose rien. Et je répète que je ne m'adresse pas aux 1% meilleurs chercheurs

2. Par exemple : la force du big data au service de la ville de demain et des transports du futur. Le problème c'est que ce genre de phraséologie triomphaliste risque de donner un faux sentiment de sécurité, de force et de pouvoir.

théoriques qui ont une vraie vocation pour leur théorie où ils sont irremplaçables. Mais j'espère que mes idées seront un jour utiles à d'autres gens, à des gens qui se sentent aujourd'hui peut-être aussi perdus que j'étais il y a 10 ans. Je m'adresse en premier, en suivant en peu la devise de ce résumé, à d'autres franc tireurs qui cherchent une autre relation avec la théorie.

J'ai déjà évoqué ces idées avec divers collègues, à la Roadef ou ailleurs. Ils étaient tous souvent d'accord sur certains points. Mais je sentais que l'essentiel de mes idées glissaient sur eux comme de l'eau sur un canard. C'est vers la fin de mes recherches que j'ai eu le plaisir de trouver un allié dans le passé de l'université d'Oxford, en la personne de Christopher Strachey (1916-1975). En fait, je ne suis pas sur que je peux l'appeler mon allié. Mais j'ai senti qu'il m'a arraché les mots de la bouche. Ce qu'il a écrit il y a 50 ans m'a ainsi bien réconforté et j'espère que cela pourra se réitérer plusieurs (maintes) fois dans l'avenir – dans l'esprit de la devise de ce résumé. Voici le texte complet affiché encore aujourd'hui à `www.cs.ox.ac.uk/activities/concurrency/courses/stracheys.html` :

> The strongly held common philosophy of the Department (previously Oxford University Computing Laboratory) is admirably expressed in the following quotation from Christopher Strachey, the founder of the Programming Research Group which now forms part of Computing Science at the Department :
>
> > It has long been my personal view that the separation of practical and theoretical work is artificial and injurious. Much of the practical work done in computing, both in software and in hardware design, is unsound and clumsy because the people who do it have not any clear understanding of the fundamental design principles of their work. Most of the abstract mathematical and theoretical work is sterile because it has no point of contact with real computing. One of the central aims of the Programming Research Group as a teaching and research group has been to set up an atmosphere in which this separation cannot happen.

©Daniel Porumbel, version révisée, Anno Domini 2022

Références

[1] Daniel Porumbel, Prize-Collecting Set Multi-Covering With Submodular Pricing, *International Transactions in Operational Research*, 25 (4) :1221-1239, 2018

[2] Daniel Porumbel, Demystifying the characterizations of SDP matrices in mathematical programming, report technique CNAM, `http://cedric.cnam.fr/~porumbed/papers/sdp.pdf`

[3] Daniel Porumbel, D'un « petit détail » d'implémentation vers un manifeste sans limite (sur l'éloignement entre la programmation et la théorie), `cedric.cnam.fr/~porumbed/soft/`