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

Daniel Porumbel, CEDRIC-CNAM

> Toute vérité qu'un homme découvre et toute obscurité qu'il arrive à élucider intéressera un jour un autre esprit pensant ; peu importe quand il la comprendra, cela le réjouira et le consolera. C'est à un tel homme que nous parlons, tout comme ceux qui nous ressemblent nous ont parlé, et ont été notre consolation dans ce désert de la vie.
>
> Arthur Schopenhauer

Cette communication porte sur les difficultés qui apparaissent sur le point de contact entre la programmation et la théorie. C'est sur ce point de contact que la courroie de transmission (de la force intellectuelle) perd une borne partie de son énergie lors de la résolution d'un problème d'optimisation. Il n'est pas toujours si facile de traverser la frontière entre le pays de la programmation et le pays de la théorie ; qu'est-ce qui fait que la vie d'un travailleur frontalier est plus difficile qu'il n'y paraît à première vue ?

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. Par ailleurs, les gens les plus attachés à la théorie ont une compréhension très approximative de la programmation et du génie logiciel ; 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 vue reste au pouvoir, la communauté sera fracturée :

(a) D'un coté, des sommités académiques qui n'ont jamais vu un algorithme en exécution 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. J'ai peur qu'ils se bornent souvent à essayer de tout comprendre en gros. 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 leur jugement 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 jeunes – l'esprit de synthèse et les bonnes grilles d'analyse pour déconstruire certains pièges du système. Cela les empêche d'avoir une vision globale très correcte sur la place et la valeur de leur travail dans un contexte scientifique plus général.

Ces idées non-techniques se projettent de milles façons sur le plan technique. On doit souvent choisir (a) ou (b). Beaucoup pensent que la théorie est si noble que la programmation doit rester une affaire secondaire par rapport à la « recherche fondamentale ». Je pense que 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%. Ils évitent souvent les questions d'implémentation pour des raisons très prosaïques : il faut prendre des instances, lancer des tests, se replonger dans le code à chaque révision ; bref, c'est trop fastidieux. En conséquence, beaucoup se disent : je vais chercher (une niche dans) une théorie. Et si j'arrive à bien m'entendre avec les quelques dizaines de chercheurs qui travaillent sur cela, ça sera que du bonheur. Et je vais pouvoir aussi appeler mon travail « recherche fondamentale ».

J'étais à deux doigts de tomber dans ce 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 a l'air chouette, on a pu gagner le prix Fulkerson (en 2003) pour des travaux là dedans, tout est très noble ; j'aurai du pain sur la planche toute ma vie quoi qu'il se passe ». Mais, non, c'était un piège ! Un jour j'ai compris que j'étais inconsciemment à la recherche d'une planque, c.à.d., une niche (lisez : une place au soleil) 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 compris qu'une autre loi régira ma vie : ceux qui choisissent une planque finissent au placard (de leur discipline au sens large).

Cela m'a poussé à éviter les niches 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 nombre 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). Je ne veux donc m'éloigner ni de la théorie ni de la programmation. *Un travail issu d'un cerveau composite, où une personne fait 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 mène à une société *fracturée et refroidie* qui évolue comme un enfant dans 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, beaucoup plus que de nombreuses niches théoriques.* Ceci n'est pas un conseil à tout-và pour tout le monde. C'est juste une idée qui sera, je l'espère, un jour utile à tous ceux qui se sentent aujourd'hui aussi perdus que j'étais il y a 10 ans. Je répète que je ne m'adresse pas aux 1% meilleurs chercheurs théoriques qui ont une vraie vocation pour leur théorie où ils sont irremplaçables. Je m'adresse, en suivant en peu la devise de ce résumé, juste à d'autres franc tireurs qui cherchent une autre relation avec la théorie.

En conformité avec la tradition Roadef, ce document de 2 pages n'est qu'un résumé d'un travail plus complet. Vous trouverez une version élargie de 3-4 pages à `cedric.cnam.fr/~porumbed/roadeffull.pdf` ou une argumentation d'une dizaine de pages à [3].

J'ai déjà évoqué ces idées avec diverses confréres, 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 d'une des meilleures université du monde (Oxford), en la personne de Christopher Strachey (1916-1975). En fait, je ne peux peut-être pas dire que c'est mon allié, mais j'ai été réconforté par les mots ci-dessous qu'il a écrits il y a un 50 ans. J'espère que cela pourra se réitérer plusieurs (maintes) fois dans l'avenir, dans l'esprit de la devise de ce résumé.

> 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.

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/`