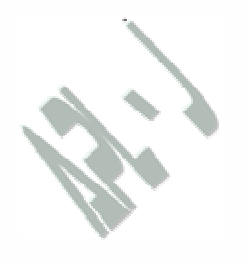
Historique du Quinto

par Gérard A. Langlet

ou

un exemple d'amélioration du schmilblic par Compression Algorithmique



La Compression algorithmique consiste, étant donné un problème quelconque, à réaliser les actions suivantes:

a) Trouver une solution au problème et écrire le programme ou l'ensemble de programmes (fonctions définies en APL, sous-programmes ou procédures en FORTRAN, PASCAL, C++ et tutti quanti) sans bogues, résolvant le maximum de cas qu'englobe le problème dans sa définition.

b) Itérer, même si cela prend des mois ou des années, la procédure suivante, indépendante du problème, et déjà décrite, bien avant la naissance de l'informatique, par Nicolas Boileau (Art Poétique, 1674) :

> *Cent fois sur le métier, remettez votre ouvrage,*
> *Polissez-le sans cesse et le repolissez.*

Traduisons en langage clair et pragmatique : il s'agit de ne jamais se contenter de l'état actuel d'un programme - appelons-le "schmilblic", mais d'intervenir régulièrement pour essayer de l'améliorer.

La règle sera, en gros, que la version suivante d'un programme devra, si possible, permettre de traiter des sous-cas du problème non envisageables à l'aide de la version courante (par manque d'espace, par lenteur ou coût trop élevé). Il n'est pas interdit de chercher en outre à réduire :

- la taille du programme lui-même (encombrement),
- la taille maximale intermédiaire de la mémoire nécessaire (surtout si le programme est dynamique, cas avec APL) ,
- le temps d'exécution, même pour les *petits* cas.

Pour parvenir à ces fins, TOUS les moyens (intellectuellement honnêtes) seront bons - on conseillera notamment d'apprendre APL rien que pour s'habituer à penser en vecteurs, en tableaux (et, si possible, au parallélisme, dans un futur proche).
Durant des années, j'ai appliqué cette méthodologie *réductionniste* à l'ensemble des programmes et des sujets abordés. Après plus de 20 d'une telle pratique, il avait surgi une question, à la fois physique et philosophique, somme toute triviale:

Un programme, comme un gaz *parfait*, est-il infiniment compressible ?

On peut se douter que la réponse est NON.

Alors, on pouvait se proposer comme objectif, lorqu'on effectue de la recherche, de trouver le(s) noyau(x) algorithmique(s) incompressible(s), et de répondre en parallèle à la question *subsidiaire* :

Si un programme P a été réduit en P puis en P puis en P et si un programme Q (n'ayant apparemment rien à voir avec le précédent) a été réduit en Q puis en Q puis en Q, qu'y-a-t-il de commun entre P et Q ?

Notes. Il vaut mieux que le problème posé corresponde à quelque chose d'utile.

Il importe, pour une meilleure crédibilité, que le problème soit posé par un tiers et non par soi-même.

Ce numéro du journal essaye de proposer une sorte de passage depuis l'ère de l'Intelligence Artificielle à celle de l'Intelligence Naturelle; nous rappellerons qu'un périodique tel que Les Nouvelles d'APL est fait pour apporter et pour recevoir de l'information (de et à toute une communauté passionnée d'améliorations, sympathique, internationale - même mondiale, et d'un niveau mathématique nettement supérieur à celui des programmeurs moyens; c'est peut-être pour cela aussi qu'APL ne peut connaître le succès d'un simple BASIC). Dès 1989, j'avais soupçonné que la réponse aux questions posées plus haut s'écrivait ≠\ (et ne pouvait donc pas être déduite aisément par des chercheurs en mathématiques, en physique ou informatique, ne connaissant pas APL). Même dans la sphère APL, cet idiome surprenant restait assez peu connu : il est absent, ou à peine mentionné, dans la plupart des manuels et livres, quels que soient leurs auteurs.

D'après [Br88], l'IA couvre la résolution des puzzles. C'est bien ce que pensait aussi JJ. Girardot, lorqu'il soumit, dans le numéro 6 [JJG93] le puzzle appelé "Inversion Diabolique" et proposa un logiciel en APL90 "*qui analyse les coups du joueur (et les siens propres) et acquiert ainsi une connaissance progressive des configurations; en tant que joueur, il devient de plus en plus difficile à battre...*").

Alors, je me dis que résoudre le puzzle en se servant des propriétés de ≠\ prouverait la puissance des concepts que je cherchais à faire admettre - très difficilement - par ailleurs.

Je me mis à l'œuvre, compte tenu des résultats obtenus par la recherche sur la compression algorithmique, et dès le numéro 7, je pus donner une solution directe - avec laquelle l'ordinateur ne peut jamais perdre s'il joue en premier - cette solution se résume à une demi-ligne d'APL-ISO (Lan93). La résolution du même problème, appelé MAGIC en Angleterre, fit aussi l'objet d'un article de ma part (The Fractal Magic Universe) publié dans VECTOR en juillet 1993. Pour les lecteurs connaissant le

livre, en français, de Maurice Dalois, relatif à APL*PLUS PC (d'ailleurs toujours disponible, avec sa disquette, auprès d'Uniware), signalons que le jeu appelé « MERLIN » au Québec est encore ... une variante du même jeu.

C'est alors qu'intervint Michel Dumontier, dans le numéro 8 avec le jeu du Quinto, dont il donnait la solution pour N ¹ 3 5 en corrigeant les aberrations du domino par une fonction capable d'inverser *numériquement* une matrice à déterminant nul. Michel affirmait que ce Quinto était un "super-puzzle" et que l'inversion matricielle binaire - initialement utilisée pour résoudre l'inversion diabolique avant que l'on trouve, par symétrie, une solution évitant l'inversion - ne saurait le résoudre dans tous les cas. Effectivement, si on en reste au jeu de JJG, l'inversion diabolique n'oblige pas à traiter des matrices à déterminant nul, simplement parce que le vecteur des masques ou inverseurs forme une base vectorielle de même rang que le jeu; mais on peut proposer des jeux plus subtils, avec des masques inverseurs redondants donc non indépendants, tels que des choix de différentes séries de ces masques permettraient de passer d'une certaine configuration de départ à un certain but. Alors, la matrice aurait un déterminant effectivement nul modulo 2. Et je n'ai jamais compris que pour résoudre un problème posé en binaire pur, il faille utiliser une méthode faite pour l'arithmétique flottante. Il manquait - et il manque toujours - en APL, la primitive « domino modulo 2 » (mais, à ma connaissance, cette primitive - ou procédure intégrée - n'existe pas non plus dans les autres langages de programmation).

Il faut dire que je m'intéressais plus à l'algorithmique de cette inversion matricielle modulo 2 qu'aux puzzles en général, mais là, un certain défi se devait d'être relevé. Michel reçut, le lendemain, par Fax, les solutions du Quinto, jusqu'à N=20, qu'il ne connaissait pas ! Dans le numéro suivant, il le mentionne et montre que la méthode du domino flottant *corrigé* ne parvient pas à trouver la solution pour N=18 (ce qui confirme la faillite de l'arithmétique en virgule flottante à cause de ses troncatures inévitables; pourtant 18 n'est manifestement qu'un tout petit nombre).

Devant organiser une table ronde ("Workshop") à APL94 sur l'algèbre binaire et ses immenses potentialités inconnues, même de la plupart des APListes, je décidai que le Quinto, royalement rebaptisé "Quin" en anglais, servirait de support thématique; grâce à l'aide de Louis Métayer, à Saclay, je pus sortir des solutions jusque N=54, et surtout montrer qu'il était possible, en APL, de calculer juste avec un micro-ordinateur, car la résolution exigeait d'inverser une matrice (à déterminant nul) de 54^4 termes sans tolérer une erreur sur un seul bit.

Je ne cherchai pas plus avant à "réduire" la résolution du puzzle, ayant fait remarquer, aussi bien dans les Nouvelles d'APL que dans APL-CAM Journal, dans l'article en anglais destiné à la table ronde, que les matrices à inverser étaient des matrices-bandes diagonales à motifs répétitifs, donc très creuses, pour lesquelles il suffirait d'adapter modulo 2 les algorithmes en vigueur pour les matrices numériques

(Cholevsky, Lanczos etc...) si l'on voulait réduire drastiquement la dimension des matrices à traiter (donc le problème, donc le temps consommé).

L'important est qu'avec l'inversion généralisée, on pouvait imaginer des problèmes avec des règles très compliquées (un clic sur une case pouvant inverser un nombre quelconque de cases un peu partout, et un clic sur une autre case inverser un jeu de cases sans rapport avec le premier, etc...) et qu'il faudrait alors bien inverser une matrice de dimension N^2 par N^2 dans le cas général pour voir s'il existait alors des solutions. En outre, le damier peut ne pas être carré mais prendre une forme quelconque; il peut même se composer d'éléments (sous-damiers) non connexes, avec des cases ne servant à rien.

Le Quinto était presque oublié lorsque je reçus de St Pétersbourg fin décembre 1994, un fax bizarre, manifestement chiffré (pour un profane), mais dans lequel je reconnus (et vérifiai) qu'il s'agissait de la solution du Quinto 100. Les explications de l'auteur, assez laconiques, me permirent néanmoins d'identifier l'émetteur et de comprendre comment ce zélé jeune homme (voir sa photo dans le numéro 14 p. 110 - une page prédestinée pour un as du binaire) avait opéré, très intelligemment et matriciellement. Je lui demandai de bien vouloir transformer son Fax en article, en lui proposant de le publier en France, en français et en primeur. Les délais d'écriture, de correction et de traduction ne nous permirent pas d'inclure ledit article dans le dernier numéro 14, d'autant plus qu'aucune fonction APL n'était jointe au premier envoi.

L'article d'Ilya montre clairement qu'un *schmilblic* peut en cacher un autre, et qu'avec APL mis entre les mains des jeunes - ce qui se pratique en Russie et pas assez chez nous - nous n'avons pas fini de comprimer les algorithmes.

(Déjà, il n'apparaît presque plus autre chose que le symbole ≠, comme prévu, et les boucles ne sont que des propagations...).
Mais nous allons profiter de la circonstance - tout à fait fortuite - pour répondre en toute impartialité, par exemple à M. Henriod, à propos de l'efficacité de l'utilisation d'APL étendu (indépendamment du constructeur).

En outre, lors de la première réunion du Bureau d'AFAPL à peine élu (le 22 mars 1995), il fut convenu que l'un des sujets à aborder lors de la prochaine demi-journée d'exposés et de débats, concernerait justement les avantages et inconvénients de la programmation en APL étendu par rapport à celle respectant l'APL-ISO, normalisé depuis 1989, en bref l'APL tout court.

J'avais écrit à Ilya Mironov que nous nous efforcions (sauf cas particuliers) de publier des listes de programmes de préférence en APL-ISO, d'abord parce que c'est la seule norme actuellement en vigueur (et à peu près respectée), ensuite parce que cela permet à tout lecteur de se servir des fonctions publiées, quelle que soit l'implantation à

laquelle il a accès, mais surtout parce que c'est souvent beaucoup plus efficace, alors que les expressions APL nécessaires ne sont pas nécessairement plus longues.

Autrement dit l'utilisation, apparemment simple, de symboles tels que ⊂ ⊃ et ¨ , peut coûter cher si on ne la pratique pas à bon escient (sans essayer les équivalents vectoriels ou matriciels dans lesquels APL excelle depuis toujours).

A preuve ce qui suit.

Reprenons les deux fonctions IGEN et IQUIN de l'article d'Ilya Mironov:

Construction de la ligne N+1 à partir de la ligne 1 (argument X):

```
    ∇ X←IGEN X;D;I;Z;⎕IO
[1]   D←⍴Z←X∧I←~⎕IO←1
[2]  IT:X←~(Z←X)≠Z≠(1↓X,0)≠0,¯1↓X ◇ →IT⌈⍳D≠I←I+1
    ∇
```

Résolution du QUIN ou Quinto N×N. Le résultat Y est la première ligne de la solution :

```
    ∇ Y←IQUIN N;X;X0
[1]   X←⊂X0←IGEN Y←N⍴0 ◇ Y←X0 SEG⊃X≠IGEN¨⊂[⎕IO]N N⍴1,Y
    ∇
```

Note. En APL DYALOG\W standard, la syntaxe correspond à celle d'APL*PLUS II et la fonction IQUIN devient par exemple IQUIND:

```
    ∇ Y←IQUIND N;X;X0
[1]   X←⊂X0←IGEN Y←N⍴0 ◇ Y←X0 SEG↑X≠IGEN¨↓[⎕IO]N N⍴1,Y
    ∇
```

Transformons IGEN en IGENM avec un argument et un résultat M matriciels tel que chaque ligne de M (matrice non nécessairement carrée) soit un vecteur X. Cela prend moins d'une minute:

```
    ∇ M←IGENM M;D;I;Z;⎕IO
[1]   D←1↑⍴Z←M∧I←~⎕IO←1
[2]  IT:M←~(Z←M)≠Z≠(0 1↓M,0)≠0,0 ¯1↓M ◇ →IT⌈⍳D≠I←I+1
    ∇
```

Alors, la fonction IQUIN se transforme aisément en ISOQUIN:

```
    ∇ Y←ISOQUIN N;A;X0
[1]  X0←IGEN Y←N⍴0 ◇ N←N,N ◇ A←N⍴X0 ◇ Y←X0 SEG A≠IGENM N⍴1,Y
    ∇
```

Cette nouvelle fonction paraîtra familière à tout APListe ne connaissant pas encore APL2 (et s'exécutera en I-APL comme en APL*PLUS I et dans tous les APL étendus)

Puis, objectivement, mesurons les résultats dans plusieurs implantations d'APL compatibles avec APL2, à l'aide de la fonction MESURE (elle-même en APL-ISO) qui fournit comme résultat le temps en secondes mis pour exécuter son argument droit, lequel contient une expression APL (ici en TryAPL2, version gratuite, compactée et francisée):

```
     N←10 ◇ MESURE'R←IQUIN N' ◇ MESURE 'RR←ISOQUIN N' ◇ RRR
0.55
0.22
1
```

Cet intéressant résultat permet de voir que la résolution du Quinto est devenue presque instantanée (une demi-seconde pour N=10), et que le retour à la norme ISO8485 a effectivement encore divisé le temps d'exécution, dans ce cas, par un facteur 2,5.

Opérons alors systématiquement:

Expression APL :	R„IQUIN N	R„ISOQUIN N
Implantation: TryAPL2 IBM 450DX2 (486 à 33 MHz)	Temps (s) N„10 0.55 N„20 1.98 N„100 48.6	Temps (s) 0.22 0.55 10.9
Expression APL :	R„IQUIN N	R„ISOQUIN N
Implantation: APL*PLUS III (sur la même machine)	N„10 0.22 N„20 0.82 N„100 19.7 N„1000 (non mesuré)	0.05 0.22 5.4 3902.
Implantation : DYALOG APL/W (sur la même machine)	N„100 37.6 (IQUIND N)	12.85
Implantation: APL.68000 Niv.2 sur Macintosh SE 1/40	N„10 4.87 N„20 17.85 N„50 127.3 N„100 667.	2.69 9.35 87.7 608.

Sous TryAPL2 et APL*PLUS III, le gain est considérable (facteur 4); il est moindre en DyalogAPL/W et en APL.68000 Level II.

En ce qui concerne les derniers essais, il ne faut pas comparer en absolu les temps obtenus sur un Macintosh dont la vitesse ne dépasse pas celle d'un PC-AT 80286.

Mais il semblerait que l'extension, compatible avec APL2, de l'APL.68000 "Level II" soit, par rapport à l'APL-ISO ("Level I") mieux programmée que celle d'APL2 et d'APL*PLUS III.

En général, la gestion des tableaux (vecteurs ou matrices) binaires composantes de tableaux généralisés, est encore mal optimisée, en place-mémoire et en vitesse, dans toutes les implantations des APL étendus; (ce qui est une condiion nécessaire pour que les futures versions puissent devenir, d'année en année, de plus en plus efficaces)

Tout jugement comparatif doit se trouver modulé en fonction du contexte.

Exemple : Ce n'est pas parce que les mesures effectuées en APL.68000 Niveau II fournissent des temps à peu près du même ordre de grandeur pour N=100, qu'il faudrait en conclure que la syntaxe de type APL2 va redevenir plus performante pour N élevé, car les comparaisons n'ont pas été effectuées non plus dans le même environnement : Si, en TryAPL2 (une application DOS), on dispose d'une mémoire libre d'environ 250 kilo-octets, les mesures en APL*PLUS III ont été effectuées avec une mémoire libre de plus d'un Méga-octet (ce qui permet d'obtenir le résultat du Quinto 1000 pendant le déjeûner). La mémoire disponible, donnée par ⎕WA dans cet essai sur ce Macintosh n'était que de 30 kilo-octets environ - ce qui n'empêche pas d'obtenir le résultat du Quinto 100 pendant le petit déjeûner.

Or, le calcul effectué par ISOQUIN 100 fabrique *temporairement* des matrices de 10000 items, invisibles par l'utilisateur, matrices dont la présence exige une réorganisation dynamique de la mémoire *gruyérisée* de la zone de travail (en anglais ce regroupement des trous de mémoire, effectué à l'insu de l'utilisateur s'appelle "garbage collection"). Un effet, de type *relativiste*, se produit dans toutes les implantations d'APL; il est moindre lorsque la mémoire disponible est grande par rapport à la mémoire déjà occupée. En outre, d'autres effets, pernicieux, apparaissent parfois, lorsqu'une zone de travail contient beaucoup d'objets, avec une table de symboles frisant l'apoplexie.

(On peut parfois rendre plus rapide une fonction aux dépens des autres, en utilisant des noms de variables locales débutant par A, B ou C plutôt que par X, Y ou Z, simplement parce que la recherche dans chaque expression interprétée trouve l'adresse en machine plus vite si la table des symboles possède une organisation - cachée à l'utilisateur - de type alphabétique. En APL.68000, les fonctions vont plus vite, en outre, si les noms ne dépassent pas trois caractères, car, autrement, la recherche du nom s'effectue par double adressage !).

La zone utilisée pour le test sur Macintosh comprenait plus de 600 fonctions avec une table des symboles - bien remplie - de l'ordre de 2000 symboles.

En conclusion, aurait-on pu prévoir, en 1993, que l'on pourrait résoudre le Quinto 100 en moins de 6 secondes en APL ? Et le Quinto 253 (entièrement affichable sur l'écran) avec TryAPL2 gratuit ? Et le Quinto 1023 avec moins d'un Méga-octet de mémoire ?

Sans l'intervention d'Ilya Mironov, il est bien clair que la réponse est NON.

Mais, on ne sait jamais, l'amélioration du schmilblic n'a peut-être pas encore atteint le nec plus ultra... Et nous serons fiers d'avoir, grâce à APL, introduit le néologisme øièëü-áëèê en russe comme un super-synonyme de Quinto, a priori extensible à TOUT problème.

Références [Voir aussi les références données par I. Mironov].

[Br88] J.A. Brown, S. Pakin & R. Polivka, *APL2 at a Glance*, Prentice Hall, N.J. (1988), ISBN 0-13-038670-7, p. 359.
[JJG93] J.J. Girardot, *L'Inversion Diabolique*, Les Nouvelles d'APL N° 6 (mars 1993), p.55.
[Lan93] G.A. Langlet, *La Rétroversion de l'Inversion Diabolique*, Les Nouvelles d'APL N° 7 (juin 1993), p. 123, et *The Fractal MAGIC Universe*, VECTOR, (B*APL*, UK), Vol. 10 No 1 (juillet 1993), pp.137-142.
[Dum93] M.J. Dumontier, *Un « super-puzzle », le Quinto*, Les Nouvelles d'APL N°7 (oct.1993). Voir aussi les numéros suivants.