APL-CAM Journal, Vol. 12, No. 1, 16 January 1990, pp 179-207. Paper presented at the Seminar "APL XI, Sofitel Diegem, Belgium, 22 November 1989". Organised by the "Genootschap Toegepaste Wiskunde en Statistiek - Technologisch Instituut K.VIV" and the "Belgian APL-CAM Users Society BACUS".

APL et l'Optimisation de Graphes Eulériens

Gérard A. Langlet
CEA/IRDI/DLPC/SCM
C.E.N. Saclay
F-91191 Gif-sur-Yvette France

Résumé

Cet article est en fait double : il présente, d'une part une méthode originale et très puissante pour la résolution du célèbre Problème du Voyageur de Commerce, et d'autre part une méthodologie de programmation en APL, simple, efficace et fort utile en général, aussi bien pour la reprogrammation des algorithmes en langage compilé que pour l'exécution en présence d'un processeur vectoriel.

Il est aussi l'aboutissement d'une démarche de pensée fort ancienne, partiellement décrite dans des articles précédents. Le choix du problème illustrant la méthodologie proposée n'est pas un pur hasard ; il s'agit même d'une gageure, tellement APL s'est trouvé jusqu'à présent déconseillé - comme assez peu efficace pour des problèmes d'optimisation, surtout fortement combinatoires, du moins dans ses versions interprétées, ce qui est encore le cas dans la plupart des implantations, notamment sur microordinateurs.

APL est un outil de réflexion et de génie logiciel Simultanément, comme ces pages vont tenter de le démontrer petit à petit. Et une comparaison finale avec un langage compilé, en l'occurrence le Pascal, va nous montrer enfin qu'APL est non seulement efficace, même certainement plus qu'on pouvait le croire jusqu'ici, à condition de s'en servir à bon escient, mais encore que les problèmes soit-disant itératifs peuvent être vus et revus d'un oeil complètement neuf.

Tout ceci constitue en fait une sorte de thèse qui pourrait s'intituler : "Optimisation tous azimuths" et que l'auteur est heureux de soumettre au jugement, si possible objectif, de la communauté scientifique internationale. Le texte est rédigé en français, il y a de très nombreuses raisons à cela…, les algorithmes et les démonstrations sont exprimées en APL, pas seulement pour économiser le papier.

Short English Abstract

This paper is in fact double: on the first hand, it describes an original and very powerful method for solving the famous Travelling Salesman Problem; on the second hand, it presents a methodology for APL programmes, which proved to be simple, efficient and very useful in general as weIl for re-programming algorithms into any compiled language as for executing APL code when a parallel processor is present.

Shorter American Abstract

This paper is in fact double: first, it describes an original and powerful method for solving the Traveling Salesman Problem ; second, it presents a methodology for APL programs, efficient for reprogramming algorithms in compiled languages and for executing APL code with a parallel processor.

Le problème dit du "Voyageur de commerce" consiste à trouver le plus court chemin circulaire reliant entre elles un certain nombre de villes sans jamais passer deux fois par la même. C'est un des plus célèbres casse-tête de la théorie des graphes ; apparemment simple, sa résolution rigoureuse s'avère en fait très difficile du fait de l'aspect combinatoire qu'il implique. En effet, si l'on désire explorer toutes les solutions possibles d'un problème à N points, il faudrait théoriquement calculer (N-1) !/2 parcours, ce qui, même avec les ordinateurs les plus puissants, reviendrait, si N est grand, à y consacrer quelques milliards d'années. L'utilité économique de se pencher sur la résolution de cas concrets apparaît néanmoins évidente à une époque où des milliers de véhicules encombrent les voies terrestres et aériennes quotidiennement. De plus, le problème est intimement lié à la connectique des circuits, donc à l'électronique et à l'informatique, aux mécanismes de reconnaissance de forme, donc à la vision et à la robotique.

On a pu montrer qu'il était illusoire, sans avoir exploré toutes les combinaisons possibles de parcours non enchevêtrés, de prétendre avoir découvert la meilleure solution. Un livre entier a été récemment consacré au sujet (1), sans apporter toutefois du nouveau quant à la résolution théorique. Heureusement, en réalité, on a souvent besoin de trouver rapidement une solution assez proche de l'optimum et cela assure déjà de substantielles économie de fils électriques, de carburant et de temps.

Le Rôle d'APL

Des dizaines d'équipes de par le monde ont mis au point des heuristiques, c'est-à-dire des mécanismes ad hoc pour trouver, si possible rapidement, des parcours parmi les plus courts - cf. la bibliographie de (1). Les mathématiciens, depuis Euler, se penchent sur le problème presque autant que sur la génération des nombres premiers ; les physiciens et les chimistes s'y sont mis, par le biais de la thermodynamique en proposant par exemple la méthode du "recuit simulé" (2).

Que viendrait donc faire APL dans cette galère ? Et pourtant… "Elementary, dear Dr Watson" (3). D'aucuns ne prétendent-ils pas qu'APL serait un outil de réflexion ? "APL as a tool of thought" a même fait l'objet du thème du plus récent congrès mondial de New York (août 1989).

Gardons l'outil en réserve et réfléchissons :

Dans chaque image perçue par nos yeux à raison de milliards de points colorés par seconde, nous reconnaissons à peu près instantanément des milliers de contours et de formes, de plus en relief et souvent en mouvement. La lecture d'une seule ligne du présent texte est bien plus compliquée que la résolution du problème du voyageur de commerce. Alors ? Seulement voilà : notre cerveau n'est pas un pauvre ordinateur. Il comporte des millions de milliards de neurones interconnectés en trois dimensions, avec des structures fractales dont le seul but ne se restreint pas au refroidissement. Il est par contre incapable d'effectuer très vite des calculs précis. Mais à quoi servent donc les

calculs d'autant plus que la nature réalise instantanément dans calcul, des millions d'optimisations sans lesquelles l'équilibre de la vie serait impossible. Ceci conduit à l'hypothèse suivante :

"Toute optimisation doit pouvoir s'effectuer rapidement au moyen d'un grand nombre d'opérations élémentaires conduisant individuellement à des résultats approximatifs."

Les spécialistes de hi "connectique" ont déjà pensé à résoudre le problème du voyageur de commerce grâce à des réseaux neuronaux et les résultats ne sont pas décevants, bien au contraire (4). Mais les réseaux que nous offre la technologie actuelle restent bien loin d'égaler la potentialité du moindre échantillon microscopique de cortex cérébral !

Une simple remarque vient à notre secours : lorsqu'un graphe est enchevêtré, comme celui de la Figure 1 présenté dans "Science et Vie Micro" (5) pour 36 villes de France, il existe toujours une solution plus courte résultant d'une simple permutation de boucle. Mais pourquoi donc engendrer des parcours comportant des boucles ? (Cette dernière réflexion est à rapprocher de la suivante, bien connue des habitués d'APL : "Pourquoi imaginer des programmes comportant des boucles quand on peut, les trois quart du temps, raisonner globalement ?"). Y aurait-il moyen de trouver très vite un parcours sans boucle, dont pourrait dériver la solution optimale, ou, tout au moins une solution acceptable, à l'aide d'opérations élémentaires, comme prescrit plus haut ? Ceci interdit évidemment tout recours aux systèmes d'équations différentielles que préconisent de nombreuses méthodes dites sérieuses, tout calcul trigonométrique et, si possible, les extractions de racines carrées. Ciel, mais où allons-nous et que reste-t-il à notre disposition ?

Intuitivement, et surtout sans référence à APL, nous pouvons supposer que notre cerveau sait effectuer des mesures approximatives de distances ou tout au moins apprécier l'ordre relatif des distances d'un point à ses plus proches voisins, le nombre de ceuxci étant assez faible dans le cas de la Fig. 1. De la même façon oue nous appréhendons très vite et sans les compter le nombre des bâtons tracés sur une feuille de papier à condition que leur nombre ne dépasse pas 6, on peut penser que la connaissance a priori du classement des distances restreint à cette limite ou même à moins - disons 5 - sera utile par la suite. La connaissance des distances exactes n'est pas indispensable comme cela a d'ailleurs été vérifié expérimentalement (Van) (6). Nous aurons donc besoin grosso modo d'opérations de comparaison, de décalage, de sommes et de tris approximatifs.

APL sait faire tout cela, hélas trop luxueusement… En effet, les calculs s'effectuent avec une précision bien trop élevée pour ce que nous voulons démontrer : une précision de 2 chiffres s'avèrerait largement suffisante à la place des 15 chiffres que nous offrent toutes les implantations. (J'en profite pour demander aux implanteurs de bien vouloir rajouter une variable du système □CP "Computing Precision", permettant d'ajuster la précision des calculs entre 2 et 15 chiffres décimaux. De la sorte, ce qui va vous être présenté ci-après pourrait s'exécuter de 10 à 100 fois plus vite et consommerait beaucoup moins de mémoire).

Une autre réflexion émanant de la technologie récente des ordinateurs vient conforter mon point de vue. On s'est rendu compte que des systèmes comportant un nombre extrêmement réduit d'instructions s'avéraient plus efficaces et plus économiques que leurs prédécesseurs, d'où la naissance de l'architecture "RISC". Tout ceci va dans le même sens, celui de la simplification. Déjà depuis deux ans maintenant, je n'utilise plus dans APL que ce qui a fait sa force, à savoir les idées initiales qui ont présidé à son apparition. De nombreux autres concepts résultant de la préhistoire de l'informatique

comme les branchements et les parenthèses ont pris du plomb dans l'aile. Les parenthèses restent utiles uniquement pour la formulation de commentaires. Exemple : (Les seules structures de données vraiment intéressantes ne sont pas celles imaginées par les informaticiens, victimes, soit de leur culture mathématique antérieure, soit des contingences encore imposées par le matériel et par la force de l'habitude. Une certaine conception Rousseauiste de l'informatique est en fait aussi importante que les considérations de K. Iverson, T. More et E. Dijkstra réunis). Les expressions et fonctions présentées ci-après seront donc écrites en APL-RISC (cf. Appendice 1), petit sous-ensemble de l'APL-ISO. N.B. : Dans tout cet article, l'origine des indices `⎕IO` est fixée à 1.

Mais revenons à nos moutons, lesquels s'impatientent. La courbe plane la plus parfaite, comme le savaient déjà les Grecs, est le cercle. Il est d'ailleurs certain que tout parcours fermé et sans boucle est une déformation gravitationnelle d'un cercle : ce type de réflexion a abouti à l'apparition de la méthode dite de "l'élastique" (7). Mais un cercle implique a priori des calculs trigonométriques beaucoup trop fastidieux. Considérer un parcours comme une déformation d'un polygone régulier simple améliorerait beaucoup les choses ; le triangle équilatéral s'exclut immédiatement à cause de la dyssymétrie qu'imposerait" un traitement en axes orthogonaux ; il faudrait travailler en axes obliques. Reste le carré, d'ailleurs plus proche du cercle que le triangle équilatéral, et qui a le mérite de conserver la symétrie du traitement par rapport à nos deux axes traditionnellement cartésiens. Mais la meilleure façon de considérer notre carré idéal sur le plan de la figure consiste non pas à le placer parallèlement aux axes, mais à le tourner de 45 degrés en le centrant sur l'origine des axes placé par exemple au centre de gravité des points. Notre méthode s'appellera donc la "mathode de l'as de carreau" ou "méthode du diamant". On définit ainsi immédiatement quatre domaines du plan qui comprennent à peu près chacun le quart des points. C'est maintenant qu'APL va donner la mesure de sa puissance. Jusqu'ici au point mort, passons en première. APL-RISC ne connaît même pas les tableaux, mais seulement les vecteurs (nous verrons pourquoi plus tard). Nous disposons juste de deux vecteurs, appelés EST et SUD, qui contiennent respectivement, avec une origine quelconque, les coordonnées X et Y (elles peuvent être inversées, cela n'a aucune importance). Nous savons effectuer des sommes, des différences et des classements approximatifs. APL sait le faire aussi, trop rigoureusement, mais contentons-nous en.

Les expressions `P←⍋EST+SUD` et `Q←⍋EST−SUD` définissent immédiatement l'ordre de rangement des villes parallèlement aux directions des côtés du diamant ; il faut maintenant mélanger astucieusement P et Q de façon à obtenir une liste circulaire V de numéros qui correspondra au parcours initial souhaité. Pour ce faire, considérons un point DISTINCT des points à relier. Ce peut être le barycentre par exemple, dont les coordonnées correspondent à la moyenne de EST et SUD respectivement - ce que tout débutant saura programmer même sans parenthèses - ou, de manière encore plus simple, un point quelconque, de coordonnées XC et YC.

L'appartenance aux quatre domaines cités plus haut est alors fixée par les conditions `i←~I EST<XC` et `j←~J SUD<YC` de telle sorte que la séquence

1	correspond à	`L1←I∧J`	puis	`V1←⌊1[Q]/Q`
2	correspond à	`L2←i∧J`	puis	`V2←⌊2[P]/P`
3	correspond à	`L3←i∧j`	puis	`V3←⍱⌊3[Q]/Q`
4	correspond à	`L4←I∧j`	puis	`V4←⍱⌊4[P]/P`

donc `V←V1,V2,V3,V4` cqfd. Voir la fonction OPTIMO en Appendice 2.

On obtient un parcours tel que celui de la Figure 2. Même pour un grand nombre de points, le temps de calcul consommé par ce processus dit d'Optimisation 0 est très faible. L'encombrement de la mémoire, surtout si l'on utilise une implantation qui gère les nombres booléens en bits, reste négligeable. À noter qu'on n'a pas jusqu'ici calculé de distance.

On voit immédiatement que des permutations d'ordre 2 ou d'ordre 3, c'est-à-dire des inversions de sections de parcours reliant 3 ou 4 villes successives améliorent considérablement la situation, du moins sur le graphe de la Figure 2. Ceci est aussi vrai en général, mais il faudra essayer des permutations d'ordre supérieur. APL va permettre de simplifier considérablement cette opération grâce à l'emploi de tests en parallèle.

Lorsque des séquences ACBD sont plus courtes que des séquences courantes ABCD, on peut effectuer les permutations cas par cas, donc en boucle. Ce sera l'habitude dans tous les algorithmes scalaires. Mais APL va permettre d'économiser un temps précieux en opérant vectoriellement.

Voir par exemple dans l'Appendice 2 la fonction DLST qui fournit vectoriellement les distances pour deux vecteurs de numéros V et W, de même longueur.

On peut aussi calculer toutes les distances possibles une fois Dour toutes et rechercher dans une table - un vecteur de dimension `N(N+1)/2`.

La fonction VDIST fabrique un tel vecteur VD. Codé en entiers, ce vecteur tient en 2.5K de mémoire pour N=36. Nous le ferons aussi par la suite dans la version en Turbo-Pascal, limitée à des cas où N ne dépasse pas 128. Mais même en APL, surtout si on pense en vecteurs et si on travaille sur une petite machine, il vaut mieux parfois répéter certains calculs rapides que d'encombrer la mémoire. Ceci a permis de mettre au point tous les algorithmes présentés ici, avec un QL Sinclair sans extension de mémoire, dans une zone de travail de moins de 20K comprenant en même temps un éditeur de fonctions et de variables en plein écran ainsi qu'un tableur graphique en couleurs, le problème posé sur les 36 villes de France (5-a) étant alors résolu en moins d'une minute jusqu'à la solution optimale. Il en a été à l'époquefaitmention dans (5-b).

Soit une séquence V quelconque. `L←V DLST 1ΦV` donne VECTORIELLEMENT les longueurs des segments successifs de telle sorte que +/L est la longueur totale du parcours. Mais on remarquera bien vite que `V DLST 2ΦV` donne le vecteur des distances deux à deux et, qu'en conséquence, `V DLST pΦV` donne le vecteur des distances p à p. Si l'on ôtait la dernière instruction de DLST, le résultat deviendrait le carré des distances, on n'extrairait plus de racines et on pratiquerait une optimisation approximative qui pourrait s'avérer suffisante. Voir en Appendice la fonction DLSTC. Raisonnons néanmoins de façon rigoureuse :

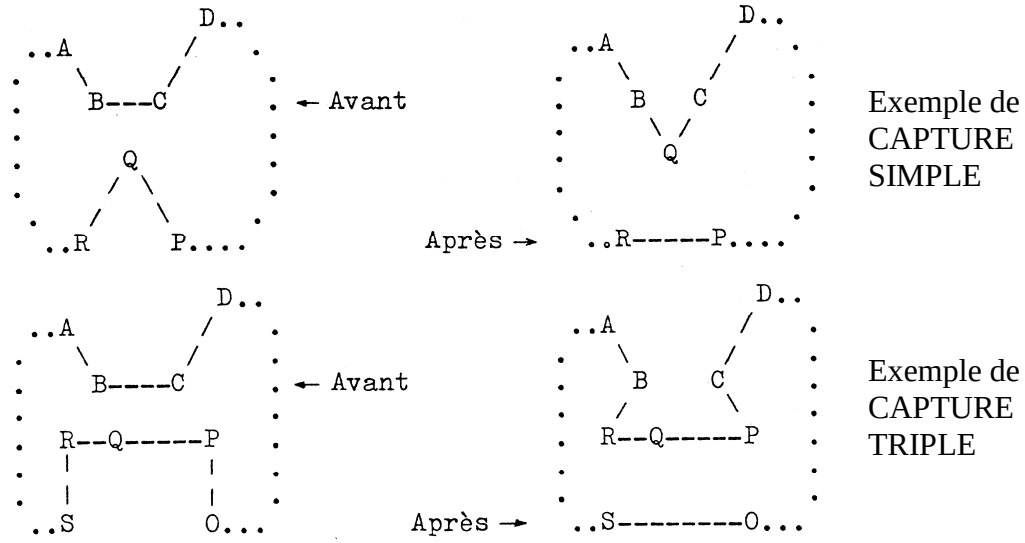
Soit P le vecteur des distances p à p. Comparons alors `I←L+pΦL` à `J←P+1ΦP`. Il est aisé de montrer que le vecteur binaire obtenu par l'expression `K←J<I` indique globalement où il faut effectuer des permutations pour raccourcir le parcours. Cela est évident pour 4 points ABC D avec p=2: ACBD est plus court que ABCD si AC+CB+BD est inférieur à AB+BC+CD. Or CB=BC donc l'expression se simplifie. Ceci se retrouve quelle que soit la valeur de p. On a pu aussi montrer qu'à partir d'un parcours quelconque tiré au hasardet donc fortement enchevêtré, des applications successives vectorielles de ces permutations, avec p variant de `2` à `⌊N÷2`, déroulent complètement le graphe. L'application de cet algorithme au résultat de l'Optimisation 0 est beaucoup plus rapide.

Dans le cas de la France (Fig. 3), il suffit même de l'appliquer avec p=3 puis 2 pour obtenir rapidement un graphe intéressant (Fig. 5), dont la longueur se situe déjà assez peu au dessus de l'optimum. La situation se modifie légèrement si on a opéré sur les carrés des distances au lieu des distances elles-mêmes. On peut aussi utiliser la fonction `DISTV` qui extrait les distances du vecteur `VD`, en lieu et place de la fonction `DLST` qui les recalcule. C'est ce que fait la fonction OPTIM1, de telle sorte qu'en cumulant : `OPTIM0` puis `OPTIM1`, on obtient en 5 secondes environ sur un Atari avec l'APL.68000 une solution à un plus de 3% au dessus de l'optimum.

Appelons cette phase l'Optimisation 1 et continuons à réfléchir.

Processus de Capture

Si un point pointe - c'est la moindre des choses - vers une autre partie du graphe, on peut envisager une capture, ou même une capture multiple :



Testons si `BQ+QC+PR` est plus court que `PQ+QR+BC` pour une capture simple, et si `BR+RQ+QP+PC+OS` est plus court que `OP+PQ+QR+RS+BC`, c'est-à-dire, en simplifiant, si `BR+PC+OS` est plus court que `OP+RS+BC` pour une capture triple. Quelle que soit la multiplicité, les tests diffèrent par un simple décalage : les fonctions de l'Optimisation 1 peuvent encore servir…

La plupart du temps, un point sera capturé par deux parmi ses plus proches voisins. Il est donc intéressant de déterminer a priori la table de proximité, restreinte, comme nous l'avons déjà mentionné, à 6 ou 7 voisins pour chaque point. Cette table `ZV` (linéarisée en APL-RISC) est donnée par :

```
ZV←⍳I←O ◇ J←G◇ G←⍳⍴V ◇ 'ZV←ZV,J↑1↓⍋G DISTV I←I+1' : ⍴V
```

Pour 36 points, elle occupe moins de 1 K. On peut la conserver sous la forme `⎕AV[T]`, si `256≥⍴V`, ce qui occupera quatre fois moins de place.

Il suffit, pour une capture simple, d'évaluer les conditions ci-dessus pour toute séquence de deux points voisins d'un troisième non adjacent aux deux premiers. Quand une capture simple est rentable, on regarde si une capture adjacente double, puis triple, etc… l'est encore plus. En pratique, même pour un grand nombre de points, on atteint rarement la capture quadruple, de sorte qu'il semble raisonnable de restreindre les tests.

Après une capture quelconque, on pourra vérifier en outre si une permutation d'ordre 2 ne fournit pas un trajet encore plus court.

A partir du parcours trouvé après l'Optimisation 1 dans le cas des 36 villes, une seule capture double, suivie d'une seule capture simple et une permutation aboutit au parcours minimal de 4441.3 km, mais on parvient aussi au même résultat, légèrement moins vite, avec 3 captures simples et une permutation.

Importance du point de départ et améliorations possibles

Nous pouvons nous douter que le cheminement de l'optimisation dépend fortement de la position du point de départ. En partant du barycentre des 36 villes, l'Optimisation 1 fournit un parcours plus long que celui obtenu avec les valeurs de XC et YC proposées dans TSP, mais l'application de captures multiples peut redonner le parcours minimal. En fait, il est toujours possible de tomber dans un minimum local, lequel sera souvent néanmoins une solution au moins aussi bonne que ce que donnerait, de manière fort dispensieuse, toute autre méthode pour le même jeu de données. Nous avons bien sar vérifié cette assertion sur les résultats proposés par d'autres auteurs, et surtout au delà de 100 points - voir plus loin.

On pourrait envisager une phase d'Optimisation 3 avec des captures en parallèle, c'est-à-dire que plusieurs points ou séquences de points se trouvent capturés simultanément par des segments différents.

De plus, lorsqu'aucune capture rentable n'est plus possible, on peut regarder si une capture non rentable, par exemple la "moins non rentable" ne produit pas un parcours sur lequel une nouvelle capture serait rentable en rattrapant la distance précédemment sacrifiée.

Cela revient presque au même que d'effectuer, sur le résultat final proposé par notre méthode, un recuit simulé. En fait, nous obtenons de bons résultats déjà très vite, surtout grâce à l'idée INITIALE de l'as de carreau. Nous savons aussi que l'optimisation serait beaucoup plus rapide s'il était possible de RÉDUIRE drastiquement la précision des calculs !

Une étrange parallèle

L'intérêt ÉNORME d'APL vient du fait qu'il est facile de tester une hypothèse nouvelle en écrivant quelques lignes simples pour la valider ou la rejeter. C'est encore plus facile en APL-RISC. On a tout sous la main ; par exemple, dans le présent article, l'ensemble des sous-fonctions de la procédure `TSP` - cf. Appendice 2 - tient dans une seule page, et un simple coup d'oeil permet d'en apprécier la cohérence, de repérer une éventuelle erreur ou de subodorer une amélioration potentielle et, souvent, de "capturer" une idée. Tiens, tiens… justement, en voici une qui naît à l'instant même : "La programmation en APL et l'optimisation du trajet du Voyageur de Commerce ne présentent-elles donc pas des points communs ?" L'Optimisation 0 correspond au tri approximatif des idées par bloc, l'Optimisation 1 à leur réarrangement "local" et l'Optimisation 2 au transfert d'idées d'un bloc à un autre ! C'est aussi exactement le processus mental habituel de composition d'un tableau, d'un morceau de musique, d'une dissertation philosophique, d'un livre… ou du présent texte.

Au cours de la rédaction de cet article, plusieurs améliorations ont effectivement vu le jour, permettant un "feedback" (en anglais dans le texte) vers les logiciels, lequel, par rapport à leur version précédente, nous a valu d'en doubler les performances qui étaient déjà très correctes.

N.B. Les expressions APL insérées à titre d'exemple ou contenues dans les listes de fonctions sont exécutables directement depuis

le traitement de texte, lui-même rédigé en APL, et leur résultat s'insère à volonté dans la page. Des mesures de temps d'exécution sont aussi prévues. C'est d'ailleurs aussi de cette manière que la traduction de la brochure (15) a été rédigée.

Performances et Pascalisation

Sur Atari ST ou Sinclair QL en APL68000, ou sur PC-XT en APL*PLUS avec coprocesseur 8087, les performances sont à peu près identiques. Si l'on possède le vecteur `VD` des distances des points 2 à 2 et le vecteur `ZV`, ordre des 6 plus proches voisins de chaque point, le programme sur les 36 villes trouve le parcours optimal en une vingtaine de secondes, sinon compter une quinzaine de secondes en plus. La totalité des logiciels et jeux de données occupe une vingtaine de Koctets. Sur PC-AT à 8MHz, compter 2 à 3 fois moins de temps. Nous avons constaté, par une programmation indépendante en APL-ISO, donc en utilisant des tableaux, que la résolution complète prenait environ 50% de temps en plus ! Si la fonction "`iw`" était remplacée par une primitive, il va de soi que nous ferions encore mieux ! Mais puisque nous nous sommes attaqués à un célèbre problème profondément itératif pour lequel APL ne semblait guère adéquat a priori, et que nous obtenons des résultats largement plus qu'honorables en langage interprété, nous avons immédiatement eu la curiosité d'utiliser un langage compilé à la mode, en l'occurrence le Turbo-Pascal, réputé rapide, lequel fonctionnant sur la même machine, pouvait permettre une comparaison de performances.

Dans le but de tester nos algorithmes sur des cas concrets publiés par des revues scientifiques, nous avons effectué une recherche bibliographique de manière à trouver des problèmes analogues, avec au moins une centaine de points, et dont la solution optimale soit déjà connue avec certitude. Nous en avons retenu deux, l'un à 100 points, donné comme exemple de la méthode dite de "l'élastique", dans Nature (7), l'autre à 120 points, concernant des villes d'Allemagne à réunir entre elles cette fois, non plus en fonction des distances à vol d'oiseau, mais des distances réelles kilométriques routières (8). Remercions ici M.Vannimenus de l'École Normale Supérieure, rue d'Ulm à Paris, de nous avoir communiqué sur disquette les 7140 valeurs nécessaires. Après des millions d'essais sur grosse machine, le parcours minimal avait pu être déterminé et, effectivement, nous n'avons jamais trouvé mieux. Noter qu'il existe 2.787292880603828E196 trajets possibles. Ce dernier problème a servi essentiellement à mettre au point la méthode dite du "recuit simulé".

Il n'existait à l'époque que la version 3 du Turbo-Pascal, mais elle satisfaisait de nombreux programmeurs. Racontons l'histoire de la mise au point du logiciel pascalisé pour l'Optimisation 0 :

Nous nous arrangeâmes pour éditer le texte-source du Pascal en gardant la faculté d'inclure des symboles APL. Ceci nous permit de mettre, sous forme de commentaire entre accolades {}, le logiciel APL. Cette première phase fut brève, vu que le corps de fonction contenait deux lignes :

```
{A←⍋EST←SUD ◇ B←⍋EST-SUD ◇ i←~I←EST<XC ◇ j←~J←SUD<YC ◇ L←I∧J
V←L[B]/B ◇ L←i∧J ◇ V←V,L[A]/A ◇ L←i∧j ◇ V←V,⌽L[B]/B ◇ L←I∧j ◇
      V←V,⌽L[A]/A}
```

La deuxième phase prit deux jours : elle consista d'abord à mettre au point, pour des vecteurs d'entiers ou réels de longueur `N≤128`, la simulation, sous forme de procédures Pascal, des fonctions primitives d'APL utilisées, à savoir dans l'ordre d'exécution : l'addition, le tri croissant, la soustraction, la comparaison, le "non", le "et", l'indexation, la réduction logique, la concaténation et le renversement. On rédigea alors les opérations sur fichiers et la partie graphique.

La troisième phase concernait l'écriture et le test de la procédure OPTIMO. Il fallut transcrire le commentaire, prévoir l'entrée conversationnelle ou le calcul des constantes XC et YC, puis programmer SANS OUBLIER LES POINTS-VIRGULES ET LES PARENTHESES. Ceci prit juste quelques minutes, et, OH MIRACLE, le programme fonctionna au premier essai !

Après un tel résultat fort encourageant, le même mode opératoire fut étendu aux Optimisations 1 et 2. Au bout de quelques journées supplémentaires passées à indenter correctement les boucles imbriquées et à peaufiner, on en vint aux mesures… pour constater que le temps d'exécution sur les 36 villes était à peu près identique en Pascal et en APL. La version Pascal, dimensionnée à 128 points, comprenait environ 800 lignes assez condensées, et il avait fallu développer quelques astuces nouvelles pour éviter un fatal débordement de la mémoire ("memory overflow") lors de la compilation.

Depuis, il est vrai, nous sommes passés à la version 5 du TurboPascal, et, en jouant sur les types de données (il existe 5 types de nombres flottants), donc sur la précision, le temps d'exécution tombe à 5.3 secondes pour les 36 villes. Nous avons donc pu vérifier aisément, en autorisant 4 puis 5 captures simultanées au cours de l'Optimisation 2, que le choix du point de départ (`XC,YC`) influait peu sur le résultat, et que l'on trouvait très souvent l'optimum à 4441 km (Fig. 6).

Il n'est pas dans notre propos de détailler ici les résultats concernant les problèmes à 100 et 120 points, qui font l'objet d'autres publications en cours. On peut néanmoins résumer les conclusions principales la méthode du "diamant" permet d'obtenir très rapidement sur micro-ordinateur (moins de 5 minutes sur PC AT), avec un logiciel d'une trentaine de Koctets, des solutions à 1 ou 2% de l'optimum, ce qui est effectivement beaucoup plus performant que toutes les méthodes classiques, et largement suffisant pour une utilisation pratique dans des problèmes de transport ou de connectique.

Avantages comparés des versions en APL et en Pascal

La version actuelle en Pascal "Turbo-TSP", "TSP" signifiant "Travelling Salesman Problem") a pour mérite de fonctionner environ 35% plus vite que celle en APL. Cela permet de gagner quelques minutes sur un problème entre 100 et 128 points. Mais ce facteur semble assez décevant lorsqu'on sait qu'APL est interprété, du moins dans les implantations que j'utilise…

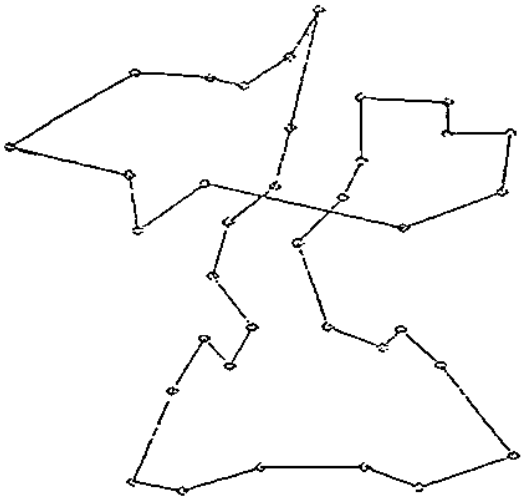
Le module Pascal compilé est indépendant du compilateur (c'est un fichier MS-DOS d'extension .COM ou .EXE). Son utilisation n'est pas sujette à redevance vis-à-vis de Borland. Par contre, l'utilisation d'une version quelconque en APL Suppose la possession de l'interprète correspondant, APL*PLUS-PC ou APL.68000. Seule la version "runtime" de ce dernier est gratuite sur Atari ST et Macintosh. Il n'existe pas

pour l'instant de versions en I-APL (très lent mais gratuit), ni en APL90 (gratuit sur Macintosh), ni en APL2 (voir à ce sujet l'Appendice 2).

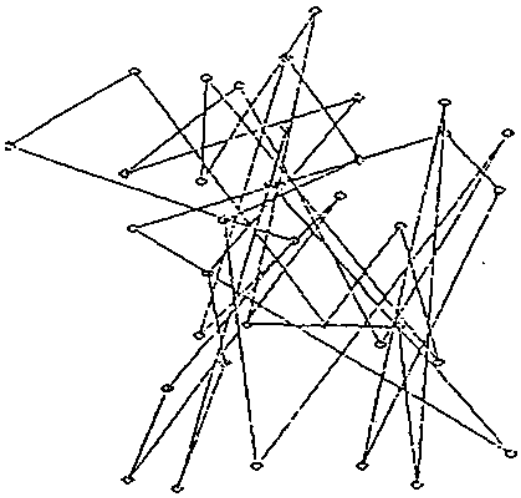
APL reste toutefois insensible aux dimensions du problème. Alors qu'une refonte de la version en Pascal au delà de 128 points semble inévitable pour différentes raisons, dont les limitations de MS-DOS et du Turbo-Pascal, on peut traiter, sans modification du logiciel, des cas bien plus importants en APL : l'exemple des Figures données à la fin de cet article, sur 162 villes des ÉtatsUnis et du Canada, passe sur un Atari 520, lequel coûte le même prix qu'un lecteur de disquettes de PC.

De toute façon, la version en Pascal n'aurait jamais pu exister sans la modélisation préalable en APL, et, même avant la réalisation de celle-ci, le fait d'avoir pensé le problème entièrement en vecteurs SUR LE PAPIER a certainement joué un rôle inestimable de clarification, aussi bien pour le problème traité que pour les perfectionnements simplificateurs de la programmation en APL, lesquels nous ont conduit à la méthodologie APL-RISC.

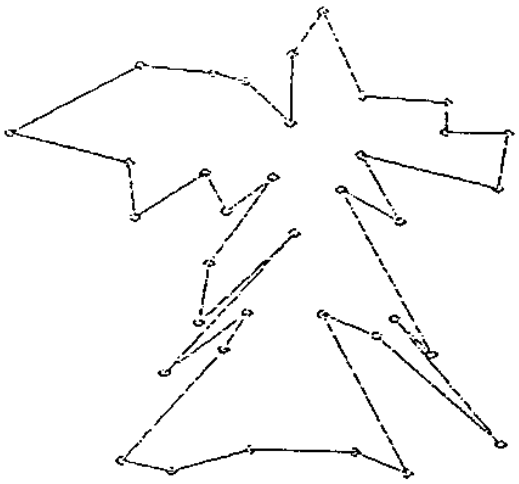
Figures



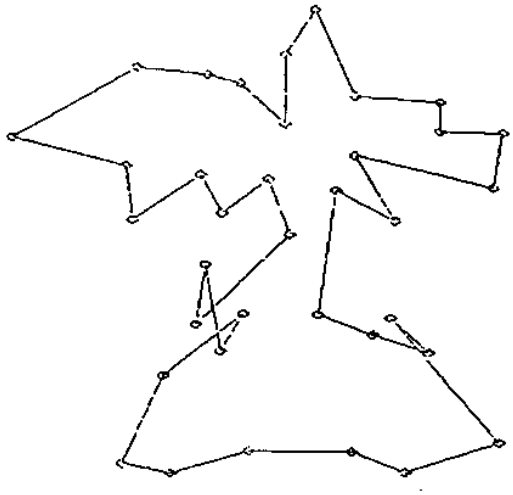
1. Parcours bouclé de 4729.8 Km tiré de (5)



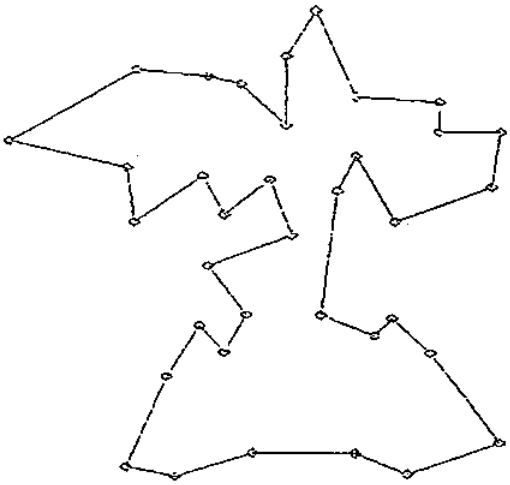
2. Parcours de 14765.1 Km dans l'ordre alphabétique



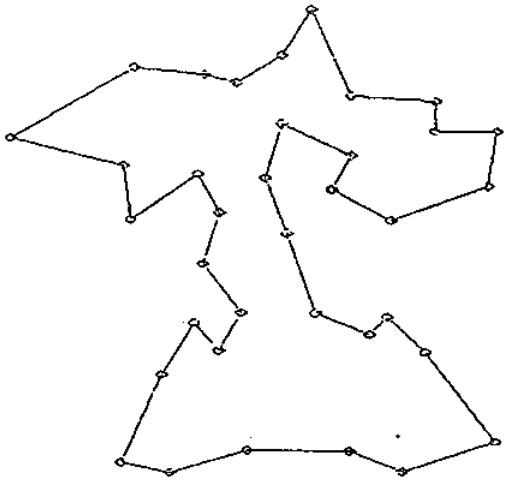
3. Parcours de 5903.2 Km
donné par 581 OPTIMO 267



4. Parcours de 5088.7 Km
donné ensuite par OPTIM1 3



5. Parcours de 4584.6 Km
donné ensuite par OPTIM1 2



6. Parcours final de 4441.3 Km
donné par OPTIM2

Appendice 1: APL-RISC

APL-RISC n'est pas - pour l'instant du moins - une nouvelle implantation. Il s'agit plutôt d'une méthodologie de programmation, simplificatrice à plus d'un titre.

Normalement, APL-RISC ne traite que des vecteurs et, éventuellement, mais nous n'utilisons pas cette possibilité ici, des vecteurs de vecteurs.

Fonctionnalités totalement abandonnées par rapport à l'APL-ISO (9) :

- les branchements, les étiquettes, les parenthèses ;
- le séparateur deux-points, dans sa syntaxe habituelle… ;
- le point-virgule, devenu inutile dans l'indexation des vecteurs (on pourrait donc facilement implanter la primitive d'indexation d'APL2);
- les primitives sophistiquées, ou, souvent, peu performantes dans la plupart des implantations, comme "encode" et "decode".

Fonctionnalités peu recommandées par rapport à l'APL-ISO:

- la concaténation sur de gros vecteurs : utiliser de préférence un remplissage par indexation après définition du résultat. Cf. (11).
- l'appel à l'interprète (exécute) pour l'exécution conditionnelle.

Fonctionnalité supplémentaire par rapport à l'APL-ISO:

- la primitive "if_while", suggérée en (14), représentée par le symbole ":" qui couvre à la fois les fonctionnalités "if", "do" et "while" des langages structurés. Exemples:

```
      '⍳5':3                          I←0 ◇ '⍳I←I+1' : 'I<3'
1 2 3 4 5                       1
1 2 3 4 5                             1 2
1 2 3 4 5                       1 2 3
```

Note: APL-RISC n'étant pas implanté, ":" est simulé par la fonction définie "`iw`". Cf. (14).

Préceptes divers:

- Une fonction s'exécute en séquence, de la première à la dernière ligne, sans JAMAIS aucun retour en arrière.
- La liste d'une fonction tient toujours en entier sur l'écran.
- Un nom de fonction commence toujours au moins par deux lettres.
- Tout nom d'identificateur comportant une seule lettre, majuscule ou minuscule, correspond à une variable qui doit être déclarée locale à la fonction primaire (cette règle constitue un soulagement).
- Les commentaires ne sont pas interdits. (Dans une implantation, le rôle de délimiteurs de commentaires ne pourrait-il pas revenir aux parenthèses COMME DANS LA LANGUE NATURELLE et la présente phrase ? Ceci aurait le mérite de faire disparaître un symbole de plus...)

N.B.: Il est précisé à titre préventif que ces préceptes ne contiennent aucun verset satanique. Tous les programmes actuellement écrits en APL-RISC (avec utilisation de la fonction "iw" à la place du symbole deux-points) restent conformes à la Norme Internationale de l'APL-ISO (9).

.
. .

Appendice 2 : Fonctions en APL-RISC relatives à l'optimisation de graphes eulériens (TSP)

```
  ∇ XC OPTIMO YC;⎕IO;I;i;J;j;L;P;Q
[1] ⎕I0←1 ⋄ P←⍋EST+SUD ⋄ Q←⍋EST-SUD ⋄ i←~1←EST<XC ⋄
  j←~J←SUD<YC ⋄ L←I∧J
[2] V←L[Q]/Q ⋄ L←i∧J ⋄ V←V,L[P]/P ⋄ L←i∧j ⋄ V←V,⌽L[Q]/Q ⋄
  L←I∧j ⋄ V←V,⌽L[P]/P
[3] ⍝ A partir des deux vecteurs de coordnonnees EST et SUD,
[4] ⍝ cette fonction fabrique le vecteur V, pernutation de
[5] ⍝ ⍳⍴V, correspondant à la séquence "as de carreau"
[6] ⍝ centrée en XC,YC et appelée "Optimisation 0".
  ∇
```

Noter que l'utilisation du diamant ⋄ ne sert ici qu'à condenser.

```
  ∇ R←A DLST B
[1] R←R×R←EST[A]-EST[B] ⋄ R←R÷A←A×A←SUD[A]-SUD[B] ⋄ R←R*.5
[2] ⍝ Calcule le vecteur des distances respectives entre les
     numéros A et B,
  ∇
  ∇ R←A DLSTC B
[1] R←R×R←EST[A]-EST[⎕8] ⋄ R←R÷A←A×A←SUD[A]-SUD[B]
[2] ⍝ Calcule le vecteur des CARRÉS des distances respectives
     entre A et B
  ∇
  ∇ VDIST;I;J;K;⎕IO
[1] ⎕IO←1 ⋄ MD←+\0,⌽1↓K←⍳⍴V ⋄ VD←⍳I←0 ⋄ 'I←1+J←I ⋄
     VD←VD,I DLST J↓K' : ⍴V
[2] ⍝ Fabrique le vecteur entier VD contenant toutes les
     distances
[3] VD←⌊100⎕OO×VD ⍝ Facultatif, mais réduit la taille de VD en
     APL68000
  ∇
  ∇ R←A DISTV B;⎕IO
[1] ⎕IO←1 ⋄ R←VD[MD[A⌊B]+1+⌈B-A] ⋄ Comme DLST; prend les
     distances dans VD
  ∇
  ∇ R←TRAJET V
[1] R←+/V DLST 1⌽V ⋄ Longueur du graphe fermé V.
[2] ⍝ Variante: R←.0001×+/V DISTV 1⌽V
  ∇
  ∇ OPTIM1 p;⎕IO;G;I;K;P;a
[1] G←⎕IO←1 ⋄ 3←'I←K⍳⌈/K ⋄ V←⌽V ⋄ V[I]←⊖V[I←⍴p]
[2] 'I←+p⌽I←V DISTV 1⌽V ⋄ a : G←∨/0<K←I-P+1⌽P←V DISTV p⌽V' :
     'G'
  ∇  Le premier : est un "if"↑         le second un "while" ↑
```

```
  ∇ OPTIM2;W
[1] N←0 ◇ 'N←V ◇ APPV ◇ OPTIM1 2 ◇ 3Φ' Km',10 3⍕TRAJET V' :
     'V⍱.≠W'  ⍝                                        "while" ↑
  ∇
  ∇ Δ← ABSENCE⍙
[1] Δ←0=⎕NC ⍙
  ∇
  ∇ TSP;⎕IO;V
[1] ⎕IO←1 ◇ V←⍳36 ⍝ FONCTION MAITRESSE REGROUPANT LES
     OPTIMISATIONS
[2] 'VDIST' : ABSENCE'VD' ◇ 581 OPTIM0 267 ◇ OPTIM1 3 ◇
     TIM1 2 ◇ OPTIM2
  ∇ ⍝    if ↑               XC↑        YC↑
  ∇ R←A sf B
[1] R←A∈B ◇ R←R/A ◇ ⍝ Equivaut à R←A~B en APL⋆PLUS, APL-SHARP
     et APL2
  ∇
```

Voici maintenant une version SIMPLIFIÉE de la fonction APPV, que l'on peut transformer ad libitum pour réaliser des captures plus complexes:

```
  ∇APPV;⎕IO;B;C;D;E;F;G;H;I;J;K;L;M;N;O;P;Q;R;S;T;U;W;X;
     Y;Z;c;e;r;z
[1] ⍝ DETERMINE SI DEUX NUMEROS SUCCESSIFS DE V APPARTIENNENT
     A J NUMEROS
[2] ⍝ SUCCESSIFS DE ZV POUR DES CAPTURES SIMPLES; TESTE AUSSI
     LA POSSIBILITE
[3] ⍝ DE CAPTURE DES 2 POINTS ADJACENTS. ECRIT EN APL-RISC...
     PUR VECTORIEL.
[4] ⍝ ICI, UN SEGMENT NE PEUT CAPTURER SES VOISINS IMMEDIATS,
[5] ⎕IO←1 ◇ J←6 ◇ G←⍳D←⍴V ◇ e←'ZV←ZV,J↑1↓⍋G DISTV I←I+1'
[6] c←'G←U[1]/1↑R ◇ K←V sf G←G,Q[1],T[1]/1↑P ◇ H←K⍳1↑B ◇
     W←H↑K ◇ V←W,G,H↓K'
[7) r←'ZV←⍳I←0 ◇ e : D' ◇ z←'ZVP←ZVP×NBP[ZV[W+I←I+1]]'
[8] 'NBP←PRMS D' : ABSENCE'NBP' ◇ r : ABSENCE'ZV' ◇
     r : 1≠⍴⍴ZV
[9] L←L×1ΦL←NBP[V] ◇ N←⍴ZV ◇ N←N÷D
[10] 'I←0 ◇ ZVP←1 ◇ W←N×G-1 ◇ z : N' : ABSENCE'ZVP' ◇ H←3×D ◇
     W←H⍴V
[11] Q←⍳I←0 ◇ H←D+¯2+⍳4 ◇ 'J←0=L[I←I+1]|ZVP ◇ J[W[H+I]]←0 ◇
     Q←Q,1↑J/G' : D
[12] L←Q≠0 ◇ H←⍳K←⍴Q←L/Q ◇ P←~W[¯1+J←D+W⍳Q] ◇ R←~W[1+J] ◇
     B←L/V ◇ C←W[1+W⍳B]
[13] J←⍴X←B,P,Q ◇ X←X,X ◇ Y←Q,R,C ◇ S←J↑Z←X DISTV V,C,Q,R ◇
     S←S-J↓Z
[14] E←⍳K←+/J←L\0>S[H]+S[H+K]+S[H+2×K] ⍝ K=NB. DE CAPTURES DE
          Q PAR BC
[15] ⍝ TEST POUR CAPTURES DOUBLES: P,Q ou Q,R OU TRIPLES:
          P,Q,R
[16] F←L/J ◇ B←F/B ◇ C←F/C ◇ 0←W[×1+D+W⍳P←F/P] ◇ Q←F/Q ◇
     S←W[1+D+W⍳R←F/R]
[17] J←⍳X←B,R,Q,P ◇ X←X,P,Q,R,B ◇ V←R,Q,C,S ◇
     S←J↑G←X DISTV Y,Q,R,S,C
[18] S←S-J↓G ◇ U←0>S[E]+S[L←E+K]+S[M←E+2×K]+S[N←E+3×K] ⍝
          R CAPTURÉ SI U
[19] J←⍴X←B,Q,P,0 ◇ X←X,Q,P,Q,B ◇ Y←Q,P,C,R ◇ S←J↑G←X DISTV
     Y,P,Q,R,C
[20] S←S-J↓G ◇ T←0>S[E]+S[L]+S[M]+S[N] ◇ c : O<⍴Q ⍝ P CAPTURÉ
          SI T
  ∇
```

REMARQUES :

- `R←PRMS N` est une fonction qui fournit les N premiers nombres premiers. Le plus simple, comme on n'en utilise ici que 36, de 2 à 151, est de posséder un vecteur tout prêt. Cette utilisation des nombres premiers à la place de l'appartenance fonctionne parfois plus vite... C'est une affaire de cas particuliers qui justifierait un autre article.
- Comme en Pascal, on définit d'abord les constantes, puis les procédures qui sont ici des chaînes de caractères; si l'on veut totalement éliminer l'usage du diamant séparateur d'instructions, ce qu'imposerait APL2, il suffit d'en faire des fonctions.
- Dans cette version, les vecteurs globaux manquants `NEP ZV` et `ZVP` sont créés s'ils n'existent pas lors de la première exécution. Les exécutions suivantes sont donc plus rapides. Les ":" sont tous des "if". On peut mettre dans une fonction séparée les captures doubles et triples, - les 6 dernières lignes - à n'exécuter que si E n'est pas vide.

`TSP` avec cette version de `APPV` peut déjà fonctionner en moins de 50 secondes malgré les boucles imposées par la vectorisation totale. Il existe des versions plus rapides et moins pédagogiques. On a aussi expérimenté une version assez voisine, qui, si aucune capture simple n'aboutit à un trajet plus court, teste néanmoins les captures multiples qui peuvent s'avérer favorables. Cette technique est appropriée pour les problèmes comportant un grand nombre de points. De même, dans le cas général, il s'avère intéressant d'utiliser, à la place de la fonction `OPTIM1`, une fonction `OPTIM1G` capable aussi de débobiner tout graphe bouclé :

```
  ∇ OPTIM1G;I;M;W;e
[1] N←¯1+⌊.5×⍴V ◇ e←'OPTIM1  I←I+1' ◇ W←0 ◇ 'Ir1 ◇ W←V ◇ e :
N' : 'V∨.≠W'
  ∇↑ while                                                  if ↑
```

Essayer par exemple : `V←⍳36 ◇ OPTIM1G ◇ TRAJET V`

`V←⍳36` est l'ordre alphabétique des villes, ce qui donne un parcours tout à fait au hasard, aussi bien que `36?36` (Fig. 2). Constater que le parcours résultant est exempt de tout bouclage et noter que sa longueur est à moins de 5% de l'optimum (4441.3 Km), indépendamment de toute autre optimisation ! Dans la plupart des cas, ce seul algorithme fournit, à partir d'un tirage au hasard, un résultat bien meilleur que celui de la Fig. 1 (5) et ce en quelques dizaines de secondes sur PC*AT .

.
. .

Comparaison de performances entre APL et APL-RISC

Cette expression est extraite de la ligne la plus lente de la fonction `APPV` (temps en secondes en APL.68000 sur Atari ST, pour `36=⍴V`).

```
  MESURE'J←⍳I←0 ◇ ''J←J,0=⌊[I←I+1]⌷ZVP'' iw ⍴V' ⍝ APL-RISC
15.6
  MESURE'J←,0=⌊∘.⌷ZVP'                          ⍝ APL
6.5
```

Il est certain que si l'on introduit des boucles, surtout simulées par la fonction définie "iw" et non pas par une primitive, la comparaison n'est pas en faveur d'APL-RISC. Mais

le but de ce dernier est de permettre la modélisation d'une idée en APL, puis la traduction vers un langage compilable, Fortran, Pascal, Cou, pourquoi pas, APL compilé. En outre, si l'on n'a pas besoin de la totalité du résultat à la fois, ou si l'on désire traiter des données plus volumineuses, - par exemple le cas des Etats-Unis pour lequel `162=ρV` et donc `26244=ρJ` - la taille de la zone de travail en APL, les limitations d'indices sous MS-DOS en général, ou encore celles des compilateurs disponibles imposent hélas très vite une programmation en boucle. Lorsque la zone de travail le permet, par exemple sur un Atari Méga-ST 4 dans lequel la taille de la zone vierge atteint 3700 K octets en APL.68000, ou dans un PS2 gonflé, avec APL*PLUS 2, ou encore sur grosse machine avec APL2 version 3 pour lequel la taille maximale théorique d'une zone frise les 128 Mégaoctets, il reste parfaitement possible d' éliminer la boucle, même en respectant les préceptes d'APL-RISC:

```
  MESURE'K←×/2ρD←ρV ◇ Z←KρZVP ◇ R←D/L ◇ J←O=R⌊z' ⍝ APL-RISC
6.6
```

On constatera alors qu'APL-RISC n'est pas plus lent qu'APL avec tableaux, ce qui indique immédiatement comment on peut programmer APPV2 de façon à diviser le temps d'exécution en gros par 2 pour l'ensemble de la procédure TSP. Il devient alors intéressant de conserver les deux versions et d'effectuer un choix dynamique automatique entre elles au moment de l'exécution, selon la place disponible en mémoire.

```
    ∇ APPV2;⎕IO;B;C;D;E;F;G;H;I;J;K;L;M;N;O;P;Q;R;S;T;U;W;X;Y;Z;
      c;e;f;r;z
[1] ⍝ MEME USAGE QUE "APPV", MAIS CETTE FOIS SANS BOUCLES DONC
      PLUS RAPIDE.
[2] ⍝ CONSOMME PAR CONTRE PLUS DE PLACE EN MÉMOIRE.
[3] ⍝ UN TRIPLE COMMENTAIRE PRÉCEDE AUSSI lES INSTRUCTIONS
      DEVENUES INUTILES
[4] ⍝ UNE FOIS CRÉÉS lES VECTEURS GLOBAUX.
[5] ⎕IO←1 ◇ J←G ◇ G←⍳D←ρV ◇ ⍝⍝⍝ e←'ZV←ZV,J↑1↓⍋G DISTV I←I+1'
[6] c←'G←U[1]/1↑R ◇ K←V sf G←G,Q[1],T[1]/1↑P ◇ H←K⍳1↑B ◇ W←H↑K ◇
    V←W,G,H↓K'
[7] ⍝⍝⍝ r←'ZV←⍳I←0 ◇ e : D' ◇ z←'ZVP←ZVP×NBP[ZV[L+I←I+1]]'
[8] ⍝⍝⍝ 'NBp←PRMS D' : ABSENCE'NBP' ◇ r : ABSENCE'ZV' ◇
    r : 1≠ρρZV
[9] L←L×1⌽L←NBP[V] ◇ N←ρZV ◇ N←M÷O ◇ K←×/Z←2ρD ◇ H←3×D ◇ W←HρV ◇
    Q←D/L
[10]⍝⍝⍝ 'I←0 ◇ ZVP←1 ◇ W←N×G-1 ◇ z : N' : ABSENCE'ZVP'
[11] J←KρZVP ◇ J←0=Q⌊J ◇ Q←ρH←4/G ◇ J[W[H+QρD+¯2+⍳4]+D×H-1]←0 ◇
    Q←1++/~∨\ZρJ
[12] H←⍳K←ρQ←L/Q←Q×L←Q≤D ◇ P←W[¯1+J←D+W⍳Q] ◇ R←W[1+J] ◇ B←L/V ◇
    C←W[1+W⍳B]
[13] J←ρX←B,P,Q ◇ X←X,X ◇ Y←Q,R,C ◇ S←J↑Z←X DISTV Y,C,Q,R ◇
    S←S-J↓Z
[14] E←⍳K←+/J←L\0>S[H]+S[H+K]+S[H+2×K] ⍝ K=NB. DE CAPTURES
      DE Q PAR BC
[15] ⍝ TEST POUR CAPTURES DOUBLES: P,Q ou Q,R QU TRIPLES: P,Q,R
[16] F←L/J ◇ B←F/B◇ C←F/C ◇ B←W[¯1+D+W⍳P←F/P] ◇ Q←F/Q ◇
    S←W[1+D+W⍳R←F/R]
[17] J←ρX←B,R,Q,P ◇ X←X,P,Q,R,B ◇ Y←R,Q,C,S ◇ S←J↑G←X DISTV
    Y,Q,R,S,C
[18] S←S-J↓G ◇ U←0>S[E]+S[L←E+K]+S[M←E+2×K]+S[K←E+3×K] ⍝
      R CAPTURÉ SI U
[19] J←ρX←B,Q,P,0 ◇ X←X,Q,P,Q,B ◇ Y←Q,P,C,R ◇ S←J↑G←X DISTV
    Y,P,Q,R,C
[20] S←S-J↓G ◇ T←0>S[E]+S[L]+S[M]+S[N] ◇ c : 0<ρ0Q⍝ P CAPTURÉ
      SI T
    ∇
```

Les seules boucles qui subsistent après cette modification dans l'ensemble de la procédure TSP sont alors parfaitement justifiées : soit elles doivent rester, car on ne voit pas comment on pourrait les remplacer par un processus vectoriel (ce sont alors des boucles vraiment itératives, nécessaires dans l'algorithme), soit elles ne s'exécutent qu'une seule fois et consomment peu de temps (cas de la génération des vecteurs globaux s'ils sont absents). Une fois la vectorisation parachevée, il paraît difficile de faire mieux, et, A PRIORI, le logiciel en APL-RISC doit fonctionner en un temps du même ordre de grandeur que celui mesuré pour sa version compilée, à peu de choses près. Cette assertion fondamentale, fruit d'une réflexion purement logique se vérifiera-t-elle en pratique ? C'est ce que nous allons voir plus loin en comparant les temps d'exécution de la version en APL*PLUS-PC et de son adaptation en Turbo-Pascal.

Autres Variantes

```
  ∇ OPTIM1R p;⎕IO;G;I;K;P;a
[1] G←⎕I0←1 ⋄ a←'I←K⍳⌈/K ⋄ V←I⌽V ⋄ V[I]←⌽V[I←⍳p] ⋄ OPTIM1R p'
[2] 'I←I+p⌽I←c DISTV 1⌽V ⋄ a : G←0∨.<K←I-P+1⌽P←V DISTV p⌽V' : 'G'
  ∇
```

C'est une version récursive de la fonction OPTIM1. Peu utile dans le cas que nous traitons, car une seule permutation d'ordre 3 puis une autre d'ordre 2 interviennent dans TSP, elle peut le devenir dans les problèmes complexes : plusieurs permutations successives améliorent parfois considérablement la situation. J'en profite pour rappeler un autre précepte : Si la récursivité apporte des satisfactions intellectuelles - voir la notation alpha-oméga, elle conduit souvent, en APL du moins, à des programmes aberrants quant au temps d'exécution, par rapport à une solution purement vectorielle. Voir à ce propos le cas de la suite de Fibonacci (10). Comme les fonctions OPTIM1 et OPTIM1R n'utilisent que la permutation la plus favorable, alors qu'il peut en exister plusieurs et que ce serait dommage de ne pas en tenir compte, du moins lorsque les dites permutations ne se recouvrent pas entre elles, la solution vectorielle se dessine déjà. Toutefois, nous ne nous engagerons pas non plus dans cette voie, car celle-ci implique de nombreux tests ainsi que des recours à toute une série de primitives, ce qui détruit la simplicité de notre démarche de pensée. Comme les distances sont précalculées, une nouvelle exécution de la fonction OPTIM1, avec la même valeur de p tant qu'un gain est obtenu, va en fait plus vite, et cet écart serait encore plus net si la primitive ":" était implantée. C'est pourquoi, à la place d'OPTIM1R, nous proposerons finalement la fonction OPTIM1B (en boucle!):

```
  ∇ OPTIM1B p;W
[1] W←0 ⋄ 'W←V ⋄ OPTIM1 p' : 'V∨.≠W'
  ∇
```

Toujours préférer la simplicité, d'autant plus que des situations analogues se produisent dans un grand nombre d'applications. En outre, si la primitive "non identique" était implantée, la notation se simplifierait encore et la vitesse du contrôle de boucle augmenterait. Plutôt également garder la mainmise sur la gestion des données que de laisser APL en engendrer des pernicieuses; cf (11).

Dans la fonction OPTIM1, il serait possible, d'effectuer un seul appel à la sous-fonction DISTV. Or, comme DISTV fonctionne très rapidement, nous avons vérifié que cette idée s'avérerait peu rentable. Il en irait tout autrement dans une petite machine avec un problème plus gros, si le vecteur VD ne pouvait pas tenir en mémoire et que nous devions utiliser DLST à la place de DISTV: le programme universel parfait n'existe pas.

Données utilisées. cf. (5-a) :

N°	NOM	EST	SUD	N°	NOM	EST	SUD
1	AMIENS	525	115	19	MONTPELLIER	655	805
2	ANGOULEME	365	585	20	MULHOUSE	310	340
3	AUXERRE	520	350	21	NANCY	8I0	245
4	BAYONNE	230	830	22	NANTES	245	405
5	BORDEAUX	305	675	23	NICE	925	785
6	BOURGES	535	425	24	ORLEANS	495	330
7	BREST	15	265	25	PARIS	525	235
8	CHERBOURG	250	140	26	PAU	320	845
9	CLERMOMT-FERRAMD	590	565	27	PERIGUEUX	410	630
10	DIJON	730	400	23	POITIERS	380	480
11	GRENOBLE	795	630	29	REIMS	655	185
12	LE HAVRE	380	150	30	RENNES	230	310
13	LE MANS	370	325	31	ROUEN	440	165
14	LILLE	580	35	32	SAINT-ETIENNE	690	600
15	LIMOGES	450	565	33	STRASB0URG	925	245
16	LYON	725	578	34	TOULOUSE	465	805
17	MARSEILLE	755	840	35	TOURS	410	390
18	METZ	810	195	36	TROYES	655	290

Les coordonnées sont en kilométres avec origine arbitraire. (Pour célébrer le Bicentenaire, nous avons tenu à traiter ces données avec un langage révolutionnaire - APL bien sûr - en décrivant ici pour la première fois une méthode et une méthodologie que nous espérons simplificatrices, donc révolutionnaires, comme le système métrique l'a été à son époque).

Mesures effectuées sur TSP en APL (avec APPV2) et
en Turbo-Pascal 5
Le pourcentage % est donné par la forMule :
100×¯1+Kilométrage÷Optimum.

Étape	PARCOURS KM	%	Secondes cumulées Atari-ST APL68000	Sun 3-160 Dyalog-APL	Bull MB-50(AT) APL⋆PLUS	TPS	⍝⍝Nachine ⍝⍝Langage
TRAJET↓36	14765.0	232.4	-	-	-	-	Au hasard
OPTIMO	5903.2	32.9	0.1	∈	0.1	∈	"DiaMant"
OPTIM1 3	5088.7	14.5	2.3	0.4	1.1	-	PerMute
OPTIM1 2	4584.6	3.2	4.9	8.7	2.2	1.3	"
OPTIM2	4470.9	0.7	13.8	1.5	4.6	3.3	Capture
OPTIM2	4441.3	0	23.0	2.4	7.3	5.3	"
Abandon	"	"	31.7	2.9	9.5	7.5	(Pas mieux)

Ces mesures n'ont qu'un caractère relatif. En APL, elles sont effectuées avec ⎕AI[2] qui, en fait, cumule le temps écoulé, dans un contexte mono-utilisateur. Sur Atari, il s'agit d'une zone de travail "effective" donc bien remplie: 3000 symboles dans la table, 1400 fonctions et 500 variables dans un contexte d'applications multiples. De plus,

divers accessoires fonctionnent simultanément, dont une horloge dateuse permanente. Malgré cela, notre objectif initial de résoudre le problème posé par (5-a) en moins d'une minute sur la machine la moins chère du marché, avec un logiciel et un jeu de données dont l'ensemble, variables intermédiaires comprises, n'occupe guère plus de 10 Koctets, est largement dépassé, et ce grâce à APL et à APL-RISC. Sur PC, l'application a été isolée et la table des symboles réduite à 100. Il reste alors plus de 400 k octets libres, ce qui donne la faculté de traiter des cas plus gros, avec graphiques; de plus, une sorte de tableur permet de déplacer les villes par noms et d'appeler aussi le module en Pascal compilé, à l'aide de □CMD. N.B. Ongagne du temps d'exécution dans de nombreuses implantations lorsque la table des symboles a une PETITE taille.

Réflexions sur ces mesures

On voit que le temps consommé par l'Opération "Diamant", processus-clé de notre méthode, est absolument négligeable, et que, très vite, on aboutit à des parcours proches de l'optimum.

Les temps consommés par la version en APL*PLUS-PC et par son adaptation en Pascal se situent, comme on l'espérait, dans la même gamme de valeurs. Les programmes ne sont pourtant pas strictement équivalents. Comme on l'a vu, Turbo-Pascal 5 travaille avec une précision moindre, ce qui, pour le problème traité, joue en sa faveur et conforte une autre de nos hypothèses initiales concernant l'optimisation en général. Noter aussi que "if" et "while" sont des primitives natives du Pascal. En outre, le programme en Pascal est plus puissant, ce qui doit le ralentir et l'accélérer à la fois, car, s'il met plus de temps pour optimiser les captures, il peut, du fait de cette optimisation, se rapprocher plus vite du but final dans le cas général:

- il utilise systématiquement la procédure "OPTIM1G" avant l'optimisation 2, et aussi avant d'abandonner, d'où le temps final élevé.
- il est étendu aux captures simultanées de 4 points et épuise plus de combinaisons de possibilités de captures. Faire la même chose exigerait une boucle supplémentaire dans "APPV2". Toutefois, ceci ne joue un rôle que dans des cas plus complexes, car il suffit, comme indiqué plus haut, de 2 captures doubles puis d'une simple dans le cas du couple XC=581 et YC=267 utilisé dans ces tests.

Le rôle du coprocesseur Intel 8087 est plus subtil quoique mineur :

- Si on le désactive, TSP-APL met 20% de temps en plus. Si, par contre, on convertit initialement les distances - le vecteur VD - en entiers, on gagne encore 10% sur les chiffres ci-dessus, avec ou sans le coprocesseur, mais la comparaison ne serait plus honnête. Arrondir les distances ne changerait rien dans le cas traité ici, car les points se situent assez loin les uns des autres. Il n'en serait pas de même s'il existait une répartition fortement hétérogène des points, avec des amas locaux, cas des villes jumelles sur la carte des Etats-Unis :
  Dallas-Fort Worth, Minneapolis-Saint Paul, etc. …(Fig. 8).
- Turbo-TSP, sauf justement pour le calcul initial des distances qui met en jeu des extractions de racines carrées, est paradoxalement plus rapide SANS le coprocesseur ! Ceci provient, comme on l'a mentionné, d'un codage des nombres réels plus condensé. C'est pourquoi on a choisi de NE PAS utiliser le coprocesseur dans toutes les mesures données ici pour la version en Pascal ; si

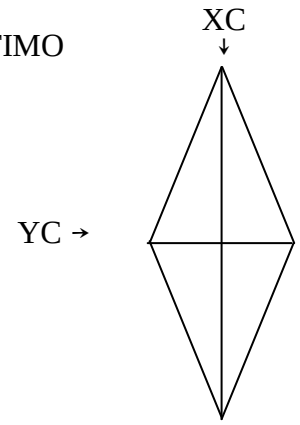
on le remettait, on obtiendrait, comme nous l'avons constaté, des temps MOINS BONS que ceux de la version en APL…

Nous allons vérifier maintenant, à l'aide de mesures, à la fois systématiques et au hasard, sur la version en Pascal, l'effet du choix des coordonnées XC et YC pour le centrage initial de l'as de carreau, afin d'étudier la robustesse de la méthode.

Les temps listés entre parenthèses correspondent aux cas où l'optimum n'a pas été trouvé. Ces cas ne sont d'ailleurs pas aussi nombreux qu'on aurait pu le croire a priori et les dernières mesures effectuées au hasard "aventureux" ont même provoqué notre étonnement légitime.

Point central		Résultat	Temps	Commentaires
XC	YC	(Km)	(s)	
581	267	4441	5.30	Identique au choix dans TSP-APL
438	533	4441	7.49	MOYENNE des points
300	250	4570	(11.70)	Essais systématiques,
400	250	4441	&.59	pour YC constant,
450	250	4441	4.07	de détermination de la
500	250	4441	6.37	plage de stabilité en XC.
600	250	4441	6.16	
700	250	4570	(11.90)	
450	450	4514	(9.17)	Essais systéMatiques,
450	400	4441	7.25	pour XC constant
450	300	4441	6.65	fixé au Minimum du temps
450	200	4441	2.25 *	des essais précédents,
450	100	4441	6.32	de détermination de la
450	0	4441	6.59	plage de stabilité en YC
200	300	4441	5.9	Essais au hasard raisonnables
300	700	4441	6.9	
700	200	4503	(17.0)	
300	500	4656	(8.5)	
0		0	4441	14.1 Essais au hasard aventureux
10000	10000	4441	16.6	
10000	-10000	4503	(14.8)	
-10000	10000	4441	15.5	
-10000	250	4441	11.0	
500	10000	4514	(16.7)	
-10000	-10000	4441	13.9	

Schéma du fonctionnement d'OPTIMO



MÉTHODE du DIAMANT
ou de l'AS DE CARREAU

Ne dirait-on pas un réticule dans un viseur, que l'on déplace sur le plan d'observation ? Noter que les dernières mesures précédentes montrent que cet instrument est particulièrement robuste et efficace, car on peut se permettre d'y voir net même dans la plupart des coins, vers 1es deux infinis du plan ! Voila enfin l'occasion de réaliser l'œil du fameux "Big Brother" de George Orwell cinq années seulement après les prévisions…

Ceci explique peut-être aussi la présence de facettes dans les yeux des insectes, et donne l'idée de paralléliser complètement le fonctionnement d'OPTIMO - voir plus loin. Sur ce principe, et sans aucun doute grâce à APL en fin de compte, pourrait-on construire des robots dont l'acuité visuelle serait sans commune mesure avec ceux que nous connaissons actuellement. On retrouve déjà le même principe en gammagraphie, toutefois avec un nombre de "viseurs" assez réduit, simplement parce qu'on n'a jamais su fabriquer des lentilles optiques capables de dévier les rayons gamma.

Comme le prévoit la théorie, on ne peut garantir de trouver l'optimum à chaque fois, mais on y parvient, dans un grand nombre de cas, avec des coordonnées initiales plus que variées. Les résultats ne sont jamais franchement mauvais. Ceci montre que des améliorations restent encore possibles, comme dans tout algorithme, mais aussi que la robustesse de la méthode s'avère remarquable.

En fait, les coordonnées 581 267 correspondent au point situé à l'est de la ville la plus au nord, et au sud de la ville la plus à l'ouest. On voit qu'en effectuant des essais systématiques horizontalement puis verticalement autour de ce point, on obtient des temps encore plus courts. Ce processus rappelle évidemment celui du balayage de l'environnement par l'œil humain en train de chercher à reconnaître des formes. Le temps extrêmement court 2.25 secondes ! - est simplement dû au fait que l'optimum est alors atteint SANS phase de capture, avec uniquement des permutations, donc trois lignes de logiciel. Vous pouvez vérifier facilement cette situation particulière, signalée dans le tableau de mesures précédent par un astérisque:

```
      450 OPTIMO 200 ◇ OPTIM1G ◇ TRAJET V
4441.3
```

On trouve d'ailleurs une certaine plage de stabilité autour de ces valeurs. Pour des problèmes plus complexes, l'expérience a montré qu'il suffit de faire des essais au centre du graphe, puis au centre de chaque étranglement du parcours obtenu, ainsi qu'au voisinage des concentrations de points, s'il en existe, pour obtenir d'excellents résultats. Mais, après un léger entr'acte consacré à un constat philosophique, nous allons indiquer comment optimiser aisément XC et YC pour automatiser complètement.

ENTR'ACTE

Réflexions annexes sur les compilateurs

Mais à quoi donc peut servir un compilateur APL ?

À peu de chose, si l'on a pris la précaution de programmer en APL -RISC. Noter toutefois qu'il sera difficile d'épuiser la controverse naissante, mais je crois sincèrement qu'il était grand temps de poser le problème.

Si le compilateur est "intelligent", c'est-à-dire capable entre autres de factoriser toutes les boucles vectorielles adjacentes, (par exemple au cours de l'exécution d'une instruction comme `-/1○V-W`, APL engendre TROIS boucles à l'interprétation, alors qu' une seule suffirait), donc de repérer un grand nombre d'idiomes, il permettra peut-être de gagner quelques broutilles, mais certainement pas de parvenir à un facteur rentable par rapport à son propre coût et au travail de réanalyse et de correction qu'il risque d'imposer en supplément.

Le principal usage d'un compilateur reste d'accélérer des programmes anciens écrits en APL laxiste, son lourd pensum consistant surtout à débobiner les accumulations de parenthèses ; il conservera en général les branchements initiaux et les dimensionnements, mais restera incapable de gérer tous les types de restructurations : on devra souvent intervenir manuellement dans les cas complexes. On peut se demander si, dans une telle situation, il ne vaut pas mieux se raccrocher à l'école philosophique du rasoir Bic, ou à l'art poétique de Boileau :

"Cent fois sur le métier remettez votre ouvrage,
Polissez-le sans cesse et le repolissez",

ce qui s'exprime laconiquement en APL-RISC par 'OUVRAGE' : 100.

.
. .

Avantages certains d'APL-RISC

Vu l'absence de parenthèses, il devient difficile d'écrire des lignes dites pornographiques. Un programme écrit en APL-RISC est plus lisible que son homologue en APL "laxiste". Il s'exécute en occupant moins de mémoire intermédiaire dans la plupart des implantations. Dans certaines d'entre elles, notamment APL.68000, il s'exécutera plus vite. On peut le traduire facilement dans un langage compilable tel que Pascal, Fortran 77 ou C, éventuellement à l'aide d'un autre programme lui-même écrit en APL-RISC. Au prix de quelques règles complémentaires qui dépasseraient le cadre du présent article, l'écriture d'un compilateur devient triviale (mais est-elle utile?). Et APL-RISC devient APL-TGV quand on l'exécute en APL2 version 3 sur un IBM 3090 muni d'un processeur vectoriel.

.
. .

Conclusion Générale

Plusieurs articles ont pour thème "APL1 vs APL2". Celui-ci fait-il partie de la série? Nous laissons au lecteur le soin de le déterminer… Mais qu'il médite aussi la dernière remarque du paragraphe précédent ; nous aurons certainement l'occasion d'en reparler.

En tout cas, l'APL-RISC est le fruit d'une expérience de plus de 15 ans de travail quotidien avec APL sur une foule d'applications de toute nature. Si l'on supprime les tableaux, d'aucuns vont souffrir, d'autres crieront à l'hérésie. Qu'ils se rappellent alors

qu'un chameau passe difficilement par le chas d'une aiguille. En fait, les tableaux ne sont pas formellement bannis du paradis des animaux, pas plus que les parenthèses. Je trouve même que les tableaux s'avèrent utiles pour la restructuration des données en vue d'une conservation ou d'une impression. L'anathème virulent concerne surtout le baudet ou le bouc d'APL (pourquoi pas le dinosaure ?). C'est pourquoi l'APL-RISC crie haro sur le GOTO.

La programmation en vecteurs purs oblige à se creuser parfois fortement les méninges (n'hésitez pas, elles sont faites pour ça), mais on trouve, dans de nombreux cas, des solutions performantes auxquelles on n'aurait guère pensé sans elle ; cela m'est arrivé aussi, entre autres applications, dans le cas de "l'apprentissage par l'exemple" : le logiciel final s'exécute environ MILLE FOIS PLUS VITE que les premières tentatives de codage en APL laxiste. Laissons le soin aux spécialistes de la question de déterminer le facteur pour le voyageur de commerce, par rapport à leur propre méthode.

À propos d'apprentissage, et bien qu'il n'existe aucun rapport, je peux souhaiter que, d'un point de vue didactique, l'APL-RISC soit aussi formateur que le latin ou les mathématiques. Aussi simple que l'espéranto, beaucoup plus facile que l'APL, et infiniment plus puissant que le Basic, il plait aux jeunes, lesquels connaissent souvent le Pascal dont il s'inspire presqu'autant que de l'APL, et c'est un bon moyen de les attirer vers les techniques vectorielles qui joueront demain un rôle capital même en micro-informatique.

Pour après-demain, c'est-à-dire lorsque les ordinateurs auront cessé de nous offrir seulement une pauvre mémoire linéaire et séquentielle, nous pourrons peut-être réinventer les tableaux, mais nous trouverons surement mieux…

.
. .

Références

(1) Lawler, E.L., Lenstra, J.K., Rinooy Khan, A.H.G. & Shmoys, D. B., livre "The Traveling Salesman Problem", J. Wiley and Sons Ltd, (1985).

(2) Kirkpatrick, S. & Toulouse, G., J. Phys. Paris, 46, 1277-1292 (1985).

(3) Doyle,C., Citation très fréquente. cf. "The Adventures of Holmes, S."

(4) Hopfield, J.J. & Tank, D.W., Biol. Cybern. 52, 3, 141-152 (1985).

(5-a) Neuville F., Science et Vie Micro, 41, (juillet 1987).

(5-b) Neuville F., Science et Vie Micro, 43, 97 (oct. 1987).

(6) Vannimenus, J., Mézard, M., J.Phys. Lett. Paris, 45, L1145- 1153 (1984).

(7) Durbin, R. & Willshaw, D., Nature, 326, 689 (avril 1987).

(8) Grotschel, M., Math. Programming Stud. 12, 61-77 (1988); Cf (1).

(9) Norme APL : ISO 8485.

(10) De Kerf, J., "Improve your Fibbing", Vector, 3, 4, 120 (avri11988).

(11) Langlet, G.A., "What is a Pernicious Loop?", APL-CAM Journal, BACUS, 11, 2, 173-176 (avril 1989).

N.B. Pour l'évolution vers APL-RISC, on peut consulter aussi :

(12) Langlet, G.A., "Vers une réforme d'APL ?", APL-CAM Journal, BACUS, 10, 2, 308-315 (avril 1988).

(13) Langlet, G.A., "Pourquoi Utiliser Préférentiellement des Vecteurs ?", Journées APL de l'AFCET "La Programmation en APL", 141-148 (avril 1982); repris dans APL-CAM Journal, BACUS, 10, 2, 316-323 (avril 1988).

(14) Langlet, G.A., "Structured Programming in APL: a Must for its Future", APL-CAM Journal, BACUS, 11, 3, 660-668 (juillet 1989).

N.B. Pour une bonne occasion d'apprendre APL simplement:

(15) Alvord,L., & Thomson,N., "An APL Tutorial", IAPL Limited, traduit et adapté en français par Langlet, G.A., BACUS Monographie 88003 (novembre 1988) - (gratuit sur demande à BACUS).

.
. .

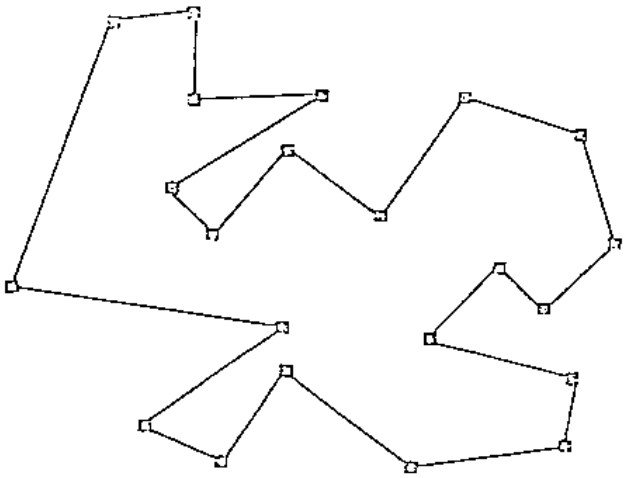
Au moment de boucler cet article, nous parvient une intéressante monographie sur les avantages du recuit simulé:

(16) Uhry, J.P., "Applications of Simulated Annealing in Operation Research", International Series of Numerical Mathematies, 87, 191-203. Birkhauser Verlag Basel 1989.

Voici un court extrait de la page 194. Nous avons agrandi le graphe original, de façon à relever les coordonnées EST et SUD des points avec une précision du quart de millimètre, soumis ce jeu à Turbo-TSP, et dessiné au premier essai le graphe donné en bas, en prenant pour XC et YC le barycentre (option automatique). Le trajet obtenu en 820 millisecondes sur Bull MB-60 (compatible PC-AT) représente 85% de l'original et est certainement le meilleur possible. Aucune capture n'est nécessaire; en optimisant XC et YC, on n'a même pas besoin de permutations d'ordre autre que 2. Ce cas est donc trivial.

2.2. Difficult problems

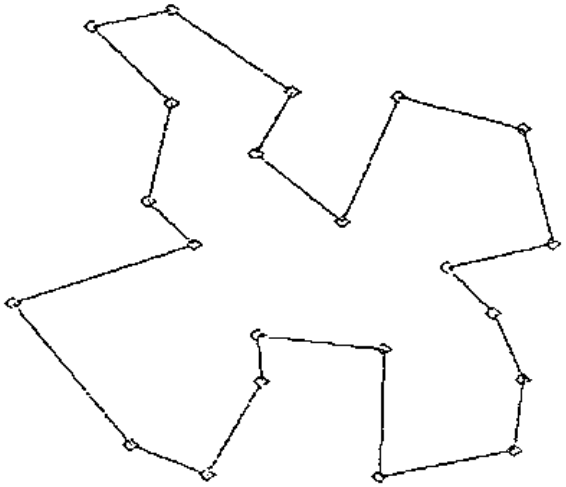
It is much harder to prove that a problem is difficult, for in order to do this, it is necessary to prove that a good method i.e. one which (is) solves the problem in every case - does not exist. For the moment no such proof exists for any "reasonable" problem. On the other hand theoreticians have established a class of equivalences, with the somewhat obscure name NP-complete, between several hundred problems for which no polynomial algorithm is currently known. An NP-complete problem has the property that if it were to be polynomial then all "reasonable" problems would also be polynomial. This provides several reasons for thinking that such a problem is intrinsically difficult.



A difficult problem

The travelling salesman problem or, how to find the shortest round trip covering a group of n towns

One of the most famous of these problems is that of the travelling salesman, who has to schedule a tour among n cities and wishes to minimize the travel distance. This problem has become the archetype of combinatorial problems (5) and was the first one studied by simulated annealing techniques (4). (For a list of NP complete problems see (3)).



Résultat Turbo-TSF

.
. .

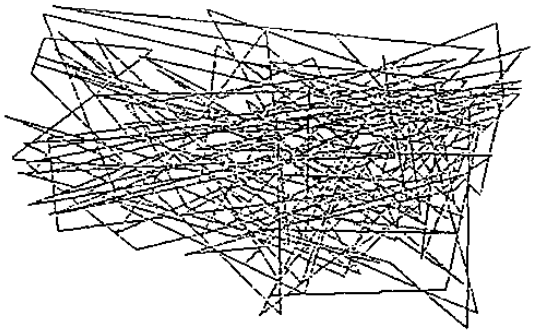
Comment baptiser une éventuelle implantation résultant d'APL-RISC De manière fort simple: si l'on prend - comme nous l'avons du reste fait au niveau des idées -l'"essence" (une des traductions possibles de "nub") de 'PASCAL' et 'APL', il ne subsiste que les lettres A, C, L, P, S. Parmi les 120 ANAGRAMMES possibles, seul émerge un mot parfaitement international, donc compris dans le monde entier : "SCALP", lequel, sémantiquement parlant, exige une réimplantation immédiate ! En français du moins, son slogan sera vite trouvé :

"SCALP, LE LANGAGE QUI DÉCOIFFE !"

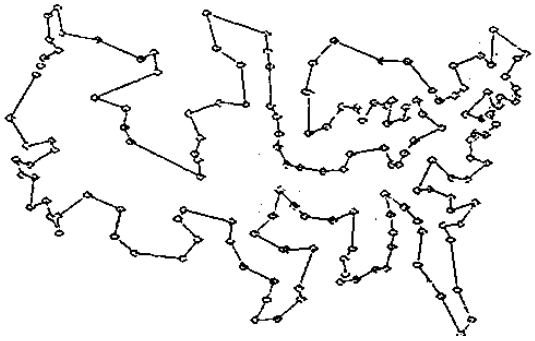
.
. .

Figures 7 & 8

Parcours au hasard : 344847.2 Km
sur 162 villes américaines



Parcours de 36970.7 Km par ISP "étendu"
On ne connaît pas l'optimum absolu



Il existe ici $3.7953525269736093 \times 10^{286}$ parcours possibles

1Φ'.Paris, le ',₸Φ3↑□TS

Paris, le 14 7 1989. cf. [9], ISO-8485:1989(F), 11.14.1, p. 139.