Tutorials in Quantitative Methods for Psychology
2012, Vol. 8(3), p. 137-150.
DOI: 10.20982/tqmp.08.3.p137

Un algorithme efficace pour la comparaison de deux moyennes indépendantes par combinatoire exhaustive
Highly efficient algorithms for the exact randomization test of the difference between two independent means

Pierre Ferland
Agence de la santé et des services sociaux de la Mauricie et du Centre-du-Québec

Louis Laurencelle
Université du Québec à Trois-Rivières

Exact randomization tests in their naïve implementation impose a computation burden that makes them generally prohibitive, forcing the researcher to choose between approximate (i.e. incomplete) randomization tests, rank-based tests or normal-family approximations. The algebra and enumeration algorithm for the comparison of two independent means are minutely analysed, and several reduction and acceleration principles are proposed. A sequential back-up mechanism turns out to be the simplest and most efficient to test the difference at the alpha (α) level, with a time-cost generally cut down by a factor better than $1-\alpha$.

Article first published in Lettres Statistiques,1988, vol. 9, p. 93-114.

Parmi les statistiques dites non-paramétriques, les épreuves permutationnelles constituent une classe à part tant par leur modalité d'exécution que par leur base théorique. Globalement, elles consistent à obtenir une estimation ou fonder une décision statistique sur la base d'une distribution des permutations des données observées, sans faire appel à une distribution parente comme c'est le cas des tests classiques fondés sur la loi normale et ses lois apparentées. Selon Bradley (1968), les épreuves permutationnelles se situent parmi les meilleurs tests, avec une performance (puissance et robustesse) comparable ou supérieure à celle des tests paramétriques.

Le principal désavantage des épreuves permutationnelles tient au fait que, la distribution-critère étant basée sur les données observées elles-mêmes, on ne peut établir de valeurs critiques ultérieurement disponibles et qu'on doit la rebâtir à chaque occasion. Cette distribution-critère, obtenue par combinatoire énumérative, varie selon le cas étudié. La comparaison de deux moyennes indépendantes, la comparaison de deux moyennes jumelées, le test d'une corrélation linéaire, etc. donnent lieu à des procédés d'énumération tous distincts: Edgington (1980) et Laurencelle (2012) en font le tour. Qui plus est, quelle qu'en soit la forme, la taille de l'énumération augmente très vite, ce qui restreint l'application de ces tests à de petits échantillons. Pour contrer ces inconvénients, les auteurs (Siegel et Castellan, 1988; Lehmann 1975) ont proposé des épreuves basées sur les *rangs* des données plutôt que sur les données elles-mêmes, ce qui permet d'établir des valeurs critiques pré-calculées et universelles. D'un autre côté, avec les données mêmes, on peut bâtir une distribution estimative, issue de permutations échantillonnées au hasard parmi l'ensemble des permutations différentes: on parle alors de combinatoire approximative, plutôt qu'exhaustive (Hope 1968; Edgington 1969; Marriott 1979; Laurencelle, 2001, 2012).

Cette étude a pour but d'examiner de près le test de différence entre deux moyennes indépendantes par combinatoire exhaustive. Nous regarderons d'abord l'algorithme de ce test, nous dégagerons certaines

Regroupements $G_1 - G_2$					
1	123 – 456	8	145 – 236	15	246 – 135
2	124 – 356	9	146 – 235	16	256 – 134
3	125 – 346	10	156 – 234	17	345 – 126
4	126 – 345	11	234 – 156	18	346 – 125
5	134 – 256	12	235 – 146	19	356 – 124
6	135 – 246	13	236 – 145	20	456 – 123
7	136 – 245	14	245 – 136		

Figure 1. Les 20 regroupements de 6 données en deux groupes (G_1, G_2) de 3 données chacun

dimensions du *coût d'exécution* de l'algorithme et nous proposerons des réductions de coût basées sur différents principes d'économie. Au-delà de la portée pratique de l'étude, nous croyons que l'analyse d'efficacité présentera certains aspects théoriques intéressants. Nous conclurons avec un programme réalisant les mérites d'un algorithme optimalisé pour la comparaison de deux moyennes indépendantes.

Le problème statistique et sa solution naïve

Supposons que nous avons évalué les sujets de deux groupes, pigés et répartis au hasard, le groupe 2 ayant bénéficié d'un traitement expérimental. En voici les données

G_1: 292 500 384 470 308 444 389 432

G_2: 468 624 367 381 530 552 552 536

Les groupes comptent 8 sujets chacun, soit $n_1 = 8$ et $n_2 = 8$. Les moyennes sont respectivement $\bar{X}_1$= 402,375 et $\bar{X}_2$ = 501,25, avec une différence D = 98,875 à l'avantage du groupe expérimental.[1]

Le test permutationnel repose ici sur le principe suivant. Selon l'hypothèse nulle (H_0), il n'existe pas de différence systématique entre les sujets des groupes, de sorte que les différences entre leurs données sont attribuables à des fluctuations d'échantillonnage. C'est donc au hasard que la donnée d'un sujet se retrouverait dans le groupe 1 ou le groupe 2, et la différence observée entre les groupes (D_o) devrait graviter près des valeurs probables des différences (D_p) issues de toutes les permutations de données. Toutes les permutations possibles et différentes de données dans les groupes correspondent ici aux combinaisons de $n = n_1 + n_2$ objets pris n_1 et n_2 à la fois, d'où $T = C(n_1+n_2, n_1) = n!/(n_1!n_2!)$. L'hypothèse pourra être rejetée si D_o, la différence observée, apparaît comme une valeur exceptionnellement forte (ou faible, selon le cas) dans la distribution des T différences permutationnelles possibles, c'est-à-dire, symboliquement, si $p_e(D_o) \leq \alpha$, où p_e désigne une *probabilité extrême*, estimée selon le rang de D_o dans la série des T valeurs de D_p, et α est un seuil de signification statistique.

Dans notre exemple, $n = 8 + 8 = 16$, et il y a $T = 16!/(8!8!) = 12870$ permutations distinctes des groupes. Il faut donc énumérer ces permutations une à une et, pour chaque permutation « p », obtenir la statistique $D_p = (\bar{X}_1 - \bar{X}_2)_p$, la comparer à D_o = 98,875 et mettre à jour un compteur selon que la différence est positive (f_+), nulle (f_0) ou négative (f_-). Au test unilatéral positif de seuil α, on rejettera H_0 si $p_e(D_o) = (f_+ + f_0)/T \leq \alpha$, tandis qu'au test bilatéral, on prendra par exemple $p_e(D_o) = [2\cdot\min(f_+, f_-) + f_0]/T \leq \alpha$. Pour nos données, par énumération complète, nous obtenons $f_+ = 207$, $f_0 = 2$, $f_- = 12661$. Pour une décision par un test bilatéral au seuil α = 0,05, nous avons $p_e(98,875) = [2\cdot\min(207, 12661) + 2]/12870 \approx 0,03232 < 0,05$; le test nous permet donc d'affirmer qu'il existe une différence sérieuse entre les deux groupes. Quant au coût d'exécution de ce test, les 205.920 additions requises, les 25740 divisions, plus des broutilles, auront consommé 65,40 secondes d'exécution du programme Basic (GWBasic, sur compatible-IBM à processeur 486, 33 MHz).

L'algorithme de base pour l'énumération des combinaisons complètes est présenté en détail à la section suivante. L'identification des facteurs de coût et l'analyse en profondeur du principe d'énumération permettront de proposer d'importants remaniements et de réduire ainsi de façon majeure le temps d'exécution de ce test.

L'algorithme de base et son coût

L'algorithme est en fait une méthode d'énumération, visant à produire les $T = C(n, n_1)$ combinaisons de groupes (G_1,G_2): chaque groupe G_k est défini par le sous-ensemble des valeurs x_i, ou des *indices* i, qui lui correspondent. Ainsi les indices {1, 2,… , n_1}, correspondant aux données {x_1, x_2, …, x_{n1}}, constituent le groupe 1 original, tandis que {n_1+1, n_1+2, …, n_1+n_2} constituent le groupe 2 original.

Prenons pour exemple deux petits groupes, de $n_1 = n_2 = 3$. Les données sont x_1, x_2, x_3, x_4, x_5 et x_6. On peut énumérer T = 6!/(3!3!) = 20 regroupements distincts de ces données. La Figure 1 présente ces regroupements, identifiés par les indices de x, en commençant avec les groupes observés. L'inspection des indices p_1, p_2, …, p_{n1} du groupe 1 fait voir une structure récursive. Cette structure est fondée sur les

[1] Pour accommoder le lecteur, les écarts-types sont respectivement s_1 = 73,988 et s_2 = 89,316, le test t, égal à 2,411, étant significatif au seuil de 5% bilatéral ($t_{14;0,975}$ = 2,145).

```
program Enum_Comb;
var n1,n2,n: integer; p:array[1..n1] of integer;

procedure comb(niveau, pos_ant:integer);
var i:integer;
begin for i:=pos_ant+1 to n2+i do
      begin  p[niveau]:=i;
             if niveau < n1 then comb(niveau+1,i)
                            else
                            begin { *** Traiter ici le regroupement défini
                               par les indices du groupe 1, p[ ] *** }
                            end
      end
end;

begin { *** Préparer les compilations statistiques ***}
      comb(1,0)
       { *** Achever et imprimer les compilations *** }

end.
```

Figure 2. Schéma de programme Pascal pour l'énumération récursive des combinaisons

deux propriétés suivantes: chaque indice démarre au-dessus de l'indice antécédent, et il plafonne à un nombre spécifique, le plafond de p_i étant $n_2 + i$. Les figures 2 et 3 présentent l'algorithme récursif d'énumération, l'un en version Delphi (ou Pascal), comportant une procédure récursive, et l'autre en version Basic.

Pour chaque regroupement de données, la compilation consiste à calculer D_p, la différence des moyennes associées à la permutation « p » obtenue.

Le coût d'exécution pour ce test sur deux moyennes indépendantes par combinatoire exhaustive, mesuré en temps ou en nombre d'opérations élémentaires, sera donc la somme de deux termes : l'un, associé à l'énumération des $C(n, n_1)$ combinaisons, et l'autre, relatif au calcul de la statistique comparative D. Le calcul de $D = (\bar{X}_1 - \bar{X}_2)$ implique n additions, 2 divisions et soustractions;[2] simplifions à n additions. Le coût global, en termes de durées d'opérations abstraites, sera donc approximativement:

$$\begin{aligned} \text{Coût (Base)} &= \text{Terme } \textit{Énumération} + \text{Terme } \textit{Calcul} \\ &= C(n, n_1) \times T_{\text{Énum}} + C(n, n_1) \times n \times T_{\text{Addition}} \\ &= C(n, n_1) \times [T_{\text{Énum}} + n \cdot T_{\text{Addition}}] , \end{aligned} \quad (1)$$

expressions dans lesquelles $T_{\text{Énum}}$ et T_{Addition} sont des durées d'opérations « élémentaires », encore à déterminer. L'équation (1) montre que le terme *Calcul* lui-même est le produit de deux facteurs, soit $C(n, n_1)$, le nombre de combinaisons considérées et $n \times T_{\text{Addition}}$, le coût unitaire de calcul de la statistique.

Le test, rappelons-le, a pour but de déterminer si le regroupement observé (G_1, G_2) est « significatif », c.-à-d. si sa statistique D_o se situe parmi les $\alpha \cdot C(n, n_1)$ ou $\frac{1}{2}\ \alpha \cdot C(n, n_1)$ statistiques D_p les plus extrêmes. Dans ce contexte, nous allons voir qu'une analyse serrée de la tâche à accomplir va permettre d'en réduire dramatiquement le coût.

Réduction du terme *Calcul*

La statistique calculée. Le test porte, on l'a vu, sur la différence de moyennes $D = (\bar{X}_1 - \bar{X}_2)$. Cette statistique D varie d'une combinaison à l'autre, mais comme il s'agit toujours des mêmes données, elle est en corrélation parfaite avec $S_1 = \Sigma_1 x_i$, la somme des données dans le groupe 1.

Soit S_1 et S_2, les sommes de données dans chaque groupe, la somme $S_T = S_1 + S_2$ est constante à travers les combinaisons. On montre aisément que $D = S_1(n_T^{-1} + n_2^{-1}) - S_T/n_2$, soit $D = A \cdot S_1 + B$, où A et B sont constants. Par un théorème classique sur la corrélation, $r(x, Ax + B) = 1$, de sorte que les distributions de S_1 et de D sont identiques, à des facteurs d'échelle près.

En outre, il est intéressant de montrer que le t de Student pour deux moyennes indépendantes varie lui aussi selon S_1, l'équation étant:

[2] Le calcul de D_p exige une soustraction, et la comparaison de D_p avec la valeur D observée en implique une autre.

```
' Programme BASIC d'énumération récursive des combinaisons
' Soit N1 et N2
'
' *** Préparer les compilations statistiques ***
'
        DIM p(N1)
        p(0) = 0 : J = 0
10 :    J = J + 1 : p(J) = p(J–1) + 1
20 :    IF J < N1 THEN GOTO 10
'
' Traiter ici le regroupement défini par les indices du groupe 1, p(1) .. p(N1)
'
30 :    IF p(J) < N2 + I THEN p(J) = p(J) + 1 : GOTO 20
        J = J – 1
        IF J > 0 THEN GOTO 30
'
'Compléter les calculs et imprimer les compilations
'
END
```

Figure 3. Schéma de programme Basic pour l'énumération récursive des combinaisons

$$t(D) = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{(n_1-1)s_1^2+(n_2-1)s_2^2}{n-2}\left(\frac{1}{n_1}+\frac{1}{n_2}\right)}}, \\ = \frac{a \cdot \mathrm{S}_1 - b}{c \cdot \sqrt{d + e \cdot \mathrm{S}_1 - a \cdot \mathrm{S}_1^2}}$$

pour $n = n_1 + n_2$ et les constantes $a = n$, $b = n_1 \cdot S_T$, $c = (n/(n-1))^{½}$, $d = n_1 n_2 \cdot \Sigma x^2$ (sommée sur les n données) et $e = 2n_1 \cdot S_T$. La corrélation linéaire entre S_1 et t n'est pas parfaite, grâce à l'influence perturbatrice du dénominateur du t. Cependant, on peut montrer que, pour toute variation de type $S_1 \pm \delta$, $\delta > 0$, le dénominateur du t fluctue avec moins d'amplitude que son numérateur ; ceci a pour conséquence que la *corrélation de rangs* entre S_1 et t est parfaite et qu'ainsi, dans la distribution permutationnelle de D, les centiles de S_1 correspondent exactement aux centiles de D et du quotient t de Student.

Ce résultat a une double conséquence. Premièrement, il suffit d'effectuer le calcul de S_1 plutôt que d'évaluer complètement la différence des moyennes. Deuxièmement, étant donné la corrélation parfaite (et négative) entre S_1 et S_2 dans un même univers permutationnel, on peut choisir de traiter le plus petit des deux groupes si ceux-ci sont de tailles inégales. On écarte ainsi deux divisions et une soustraction, mais surtout m additions seulement sont requises, en supposant $m = \min(n_1, n_2)$, soit:

$$\text{Coût } (S_m \text{ simple}) = C(n, n_1)\,[T_{\text{Énum}} \times m \cdot T_{\text{Addition}}], \quad m = \min(n_1, n_2) \qquad (2)$$

Dans le cas le moins avantageux, $m = n_1 = n_2$, le coût d'exécution est néanmoins réduit de moitié par cette simplification.

La sommation récursive. L'algorithme, comme on l'a vu, complète progressivement l'ensemble d'indices { p_1, p_2,…, p_{n1} }, correspondant aux données du groupe 1, pour permettre le calcul éventuel de la somme S; cette progression va du premier indice (p_1) jusqu'au dernier (p_{n1}). À l'image de cette progression d'indices, on peut calculer progressivement la somme cherchée en passant par des sommes partielles, basées sur les indices déjà présents: c'est le principe de la sommation récursive.

Soit S(i), la somme partielle de niveau récursif i, dans le groupe 1. Nous voulons obtenir S(n_1) = S, et l'on a S(i) = $x(p_1)$ + $x(p_2)$ + … + $x(p_i)$ = S(i–1) + $x(p_i)$. En démarrant avec S(0) = 0, la somme peut être mise à jour par une addition à chaque progression récursive, puis corrigée par la soustraction correspondante à chaque régression. La section de programme Basic, à la Figure 4, précise cette méthode.

Quel est le coût résultant de ce calcul récursif, qui à présent oblige à faire à la fois des additions et des soustractions? Soit $m = \min(n_1, n_2)$. L'étude montre que l'on manœuvre l'indice p_m, c'est-à-dire qu'on effectue S(m–1) + $x(p_m)$, un nombre de fois égal à C(n, m). Pour p_{m-1}, il varie C (n–1, m–1) fois, etc. Ainsi, le nombre de manœuvres total pour obtenir S_m est Q = C(n, m) + C(n–1, m–1) + … + C(n–m+1, 1) = C(n+1, m) – 1. Cette façon de procéder implique qu'en arrivant au niveau récursif i, on additionne d'abord $x(p_i)$, puis qu'en y revenant on le soustraie. Comptant la soustraction comme l'addition, le coût devient alors:

```
' *** Section de programme BASIC
' *** Sommation récursive de S1
        DIM(N1), X(N1+N2)
        p(0) = 0 : I = 0 : S1 = 0
10 :    J =J+1: p(J) = p(J–1) + 1
20 :    S1 = S1 + X(p(J))
        IF J < N1 THEN GOTO 10
'
' Traiter ici le regroupement défini par les indices du groupe 1, p(1) .. p(N1)
'
30 :    S1 = S1 - X(p(J))
        IF p(J) < N2 + J THEN p(J) = p(J) + 1: GOTO 20
        J = J – 1
        IF J > 0 THEN GOTO 30
```

Figure 4. Section de programme BASIC: calcul récursif de S_1, la somme dans le groupe 1

$$\text{Coût } (S_m \text{ récursif}) = C(n, m)T_{\text{Énum}} + [2\cdot C(n+1, m)-2]T_{\text{Addition}} \quad (3)$$

Ce calcul récursif de la somme S_m peut cependant bénéficier d'un allègement. Considérons que nous revenons au niveau i, après avoir exploré S(i+1), etc. À moins d'avoir atteint la limite du niveau i (= n–m+i), il faut passer à la position subséquente de ce niveau, soit soustraire par S(i) – $x(p_i)$, puis additionner par S(i)+$x(p_{i+1})$. Il est remarquable que cette double opération puisse s'écrire S(i) + [-$x(p_i)$+$x(p_{i+1})$], ou encore S(i)+d_j, où $j = p_i$ et $d_j = x_{j+1} - x_j$. Les deux opérations d'addition et de soustraction se ramènent alors à une seule opération d'addition. Remarquons toutefois que cette simplification s'applique seulement après qu'on soit installé au niveau récursif i ; pour s'y installer à neuf (suite à une progression d'indices des niveaux moins profonds), il faut faire une addition simple, suivie d'additions mixtes (= addition avec soustraction), terminées enfin par une soustraction simple, avant de remonter au niveau récursif supérieur. Le coût symbolique de ce procédé est [2·(Additions simples) + (Additions mixtes)]. Il y a 1 addition simple de niveau 1, (n–m+1) de niveau 2, C(n–m+2, 2) de niveau 3, … ,C(n–1, m–1) de niveau m ; au total, ΣC(n–m+i, i) [i = 0..m–1] = C(n, m–1). Quant aux additions mixtes, il y en a (n–m) au niveau 1, C(n–m+1, 2) au niveau 2, jusqu'à C(n–1, m) au niveau m, et ΣC(n–m–1+i, i) [i = 1..m] = C(n, m) – 1. Ainsi, le coût d'exécution, en termes d'additions et soustractions, serait ici:

$$\text{Coût } (S_m \text{ avec différences}) = C(n, m)T_{\text{Énum}} + [2\cdot C(n, m\text{-}1) + C(n, m)-1]T_{\text{Addition}} \quad (4)$$

On vérifie aussi que le total d'opérations hiérarchiques égale la somme des additions simples et des additions mixtes, soit C(n+1, m) – 1 = C(n, m–1) + C(n, m) – 1.

Pour mieux apprécier les avantages potentiels de ces différentes tactiques de calcul, nous avons évalué les fonctions symboliques de coût correspondant aux formules (1) à (4), en assignant une valeur d'unité au coût d'addition (T_{Addition}) et une valeur nulle au coût d'énumération ($T_{\text{Énum}}$). Prenons l'exemple de groupes égaux, avec $n_1 = n_2 = m = 5$. Il y a 252 combinaisons, et le coût de base d'après la formule #(1), disons $\$C_1(5,5) = 2520\cdot T_{\text{addition}}$, ou simplement 2520. Comme dans notre exemple $m = n$, $\$C_2(n, n) = \$C_1(n, n)$, soit $\$C_2(5,5) = 1260$. Pour le calcul récursif simple, on a $\$C_3(5,5) = 922$; en incluant le raffinement du pré-calcul des différences successives (mais sans ajouter le coût de ce pré-calcul), on obtient $\$C_4(5,5) = 671$. Le Tableau 1 permet de juger l'avantage lié aux deux formes de calcul récursif : on y présente les quotients de coût des deux méthodes récursives par rapport au coût du calcul simple de S_1. Le quotient pour le calcul récursif simple tend vers $\$C_3 / \$C_2 \rightarrow 4 / (m + 1)$, et pour le calcul « raffiné », vers $\$C_4 / \$C_2 \rightarrow 3 / (m + 1)$, les bénéfices croissant évidemment avec la taille du groupe-cible.

Ajoutons un dernier, petit, raffinement. Lorsque chaque combinaison est complète, on dispose de la somme permutationnelle S_P qu'il nous faut comparer à la valeur observée de la somme du groupe 1, S_1. Cette comparaison équivaut à une soustraction, qu'on peut pré-effectuer. Pour cela, plutôt que d'amorcer la sommation récursive par S(0) = 0, il suffit de l'amorcer avec S(0) = $-S_1$. En aboutissant au niveau récursif m, la statistique S(m) contiendra implicitement la soustraction de S_1.

Comme nous l'avions fait avec l'algorithme de base, nous avons testé nos idées en les incorporant dans un programme Basic et en en mesurant le temps d'exécution. Les quatre améliorations sont, rappelons-les: l'utilisation de la somme du plus petit groupe, le calcul récursif de cette somme, l'accélération du calcul récursif par un pré-calcul des différences successives, la pré-soustraction de la somme de référence. L'exemple déjà exploité comporte 16 données, avec $n_1 = n_2 = 8$. Le temps d'exécution du programme

Tableau 1. Quotients de coût des méthodes de calcul récursif de S_1 par rapport à la sommation simple, pour $n_1 = n_2 = m$ (en supposant $T_{énum} = 0$)*

m	2	3	4	5	6	8	10	12	15
(3)÷(2)	1,500	1,133	0,893	0,732	0,619	0,472	0,381	0,321	0,258
(4) ÷(2)	1,083	0,817	0,646	0,533	0,452	0,347	0,282	0,237	0,192

*Le coût de base #(1) est approximativement le double du coût de calcul simple #(2).

modifié est de 14,25 s, comparativement à 65,40 s pour le programme de base. Le quotient de coût est de 21,8 %, alors que dans nos formules nous prédisions $\$C_4/\$C_1 \approx 17{,}4$ % (= 34,7/2) au Tableau 1. Cet écart est imputable au fait que, pour construire le tableau, nous avons stipulé une durée nulle ($T_{énum}$) pour l'énumération même; en utilisant le rapport $T_{énum}/T_{addition} \approx 0{,}90$, les quotients de coût se correspondent.

Réduction du terme *Énumération*

À supposer que le calcul de la statistique utile à la comparaison des moyennes soit réduit à son expression la plus efficace, y a-t-il d'autres améliorations à espérer quant au coût d'exécution de l'algorithme de comparaison? Peut-être pas, si le but est de constituer la distribution permutationnelle complète de la statistique, puisqu'alors les $T = C(n, n_1)$ résultats différents de $D = (\bar{X}_1 - \bar{X}_2)$, ou équivalemment de S_1, sont requis. Toutefois, l'on entreprend le plus souvent cette comparaison dans le but de déterminer si la différence observée est statistiquement significative à un seuil α convenu : dans un univers où le hasard seul jouerait, à l'abri des variations expérimentales ou systématiques, est-ce que la différence D observée fait partie des 100α% différences les plus exceptionnelles? Reformulé dans le contexte de la distribution permutationnelle de notre statistique D ou S_1, le test revient à vérifier si la valeur observée de la statistique, D_o, fait ou non partie des αT valeurs permutationnelles les plus extrêmes.

Si l'on pouvait explorer seulement les αT valeurs extrêmes, tel que le laissent entendre Siegel et Castellan (1988), le test serait déclaré significatif selon que D_o fait partie ou non de ces valeurs, et l'on aurait réduit le coût d'énumération par un facteur de $(1 - \alpha)$. En fait, si l'on pouvait énumérer les valeurs permutationnelles dans un ordre strict, la réduction pourrait même être plus grande lorsque D_o s'avère significative. Soit $p_e = \min[r(D_o), T - r(D_o)+1]/T$, la *probabilité extrême* de D_o, où $r(D_o)$ est le *rang* de D_o dans l'ensemble des T valeurs possibles. Alors, par l'énumération ordonnée des combinaisons commençant par la valeur D_p la plus extrême, le nombre de combinaisons utiles au test est de $T \cdot \min(\alpha, p_e) + 1$, ce qui représente une réduction d'environ $\max[(1-\alpha), (1-p_e)]$ du coût d'exécution. Il est difficile de construire un procédé d'énumération observant l'ordre strict stipulé, de sorte que ce nombre de combinaisons, $T \cdot \min(\alpha, p_e) + 1$, représente en fait un minimum.

Plutôt que d'envisager n'énumérer que les combinaisons à valeurs les plus extrêmes, nous procéderons presque au contraire, en énumérant les moins extrêmes. Toutefois nous escomptons une réduction aussi grande du coût d'exécution, puisque l'énumération des combinaisons moins extrêmes restera *implicite*; nous n'aurons pas à les considérer une par une ni à calculer la statistique de référence. Nous allons proposer trois principes à cet effet. Quelques précisions sur l'exécution du test lui-même sont d'abord de mise.

La procédure du test ; les conditions simplifiées. Pour simplifier l'exposé, nous supposerons que nous avons affaire à une différence D_o positive, à une valeur (relativement) forte de S_1, et que nous envisageons un test unilatéral de niveau α. Pour un test bilatéral, il suffira d'exploiter un seuil de $\frac{1}{2}\alpha$. Pour que les méthodes présentées plus loin s'appliquent à une différence observée négative, il faudra *inverser* les structures orientées utilisées par nos principes d'économie. Par exemple, pour une différence négative, le tri du vecteur des $n_1 + n_2$ données observées devra se faire dans un ordre ascendant (de la plus petite à la plus grande valeur) plutôt que descendant, et le test de chaque combinaison sera affecté d'un signe inversé, c.-à-d. $D_p < D_o$ plutôt que $D_p > D_o$. Cela dit, il s'agit de déterminer si S_1 fait partie des αT valeurs permutationnelles les plus fortes. L'algorithme d'énumération a pour but d'établir les nombres de valeurs inférieures (f_-), égales (f_0) et supérieures (f_+) à la valeur observée. Le test sera déclaré significatif si $(f_+ + f_0) \le \alpha T$, ou non significatif si $(f_+ + f_0) > \alpha T$.[3]

[3] Certains préfèrent utiliser le critère « $f_+ + \frac{1}{2}f_0) \le \alpha T$ » dans lequel les valeurs égales sont réparties de part et d'autre de « l'intervalle » centré sur D_o; cette suggestion est inspirée sans doute d'une « correction de continuité », laquelle a pour but usuel d'améliorer l'approximation normale d'une distribution exacte. Comme nous disposons ici de la distribution exacte, la raison d'être de cette division des cas

Le défoncement de la zone critique. En cours d'énumération, l'un des nombres f_+ ou f_0 est augmenté de +1 à chaque fois qu'on obtient $S_p \geq S_1$. Si, quand l'énumération parvient à la permutation p, on observe $(f_+ + f_0)_p > \alpha T$, cela signifie que la valeur observée S_1 ne fait pas partie des αT valeurs les plus extrêmes, et la zone critique du test est défoncée. On peut dès lors déclarer le test non significatif, ce avant d'avoir complété l'énumération. Autrement dit, on peut cesser l'énumération des combinaisons lorsqu'on a produit suffisamment de valeurs extrêmes pour accepter l'hypothèse nulle. Ce principe peut s'énoncer comme suit:

> *Principe d'économie 1* :
>
> L'énumération des T combinaisons complètes peut être stoppée précocement lorsque la zone critique, constituée de αT valeurs égales ou plus extrêmes par rapport à la valeur observée, est défoncée.

Le gain à escompter par le principe du défoncement de la zone critique dépend largement des données traitées. Ce gain sera nul si la différence D_o (ou S_1) s'avère significative, puisqu'alors le nombre de valeurs aussi extrêmes que D_o est inférieur à la taille de la zone critique. Si la différence n'est pas significative, l'importance du gain variera plus ou moins au hasard, selon le rang de D_o dans l'ensemble permutationnel.

Soit p_e, la probabilité extrême de D_o définie d'après son rang parmi les T valeurs permutationnelles possibles, et soit $S = f_+ + f_0$ et $I = f_-$, les nombres de valeurs supérieures ou égales et inférieures, respectivement ; $S = \lceil Tp_e \rceil$. Admettons enfin que nous énumérons les permutations plus ou moins au hasard et *sans remise*. La distribution de probabilité de T', le nombre de permutations nécessaires pour obtenir $k = \alpha T$ valeurs supérieures à D_o, obéit à une loi de type Pascal, dont le terme général est:

$$(T' = k + r) = \left(\begin{array}{c} k + r - 1 \\ k - 1 \end{array} \right) \frac{S_{(k)} I_{(r)}}{T_{(k+r)}}$$

S'il y avait remise, on aurait une loi de Pascal, et l'espérance serait $\alpha T(1-p_e)/p_e$. L'évaluation de l'expression ci-dessus pour des univers assez grands (p. ex. $n_1 = n_2 = 8$ et $T = 12870$) montre que l'espérance est approximativement $E(T') \cdot \alpha T/p_e$; en fait, pour couvrir les situations où la différence observée serait significative, il faut plutôt écrire :

$$E(T') \approx \alpha T / \max(\alpha, p_e) .$$

La réduction de coût encourue par ce principe d'économie dépend du nombre T' de combinaisons requises avant le défoncement de la zone critique, s'il a lieu. Parmi les T combinaisons possibles, la proportion minimale est α et la proportion maximale, 1. Une proportion moyenne peut être estimée si on suppose vraie l'hypothèse nulle et une distribution uniforme de p_e entre 0 et 1 (ou du rang de D_o entre 1 et T). Ainsi, les nombres minimal, moyen et maximal de combinaisons à énumérer en vertu de ce principe d'économie seraient:

$$T' = \{ \alpha T, \alpha T(1 - \log_e \alpha), T \} .$$

La proportion moyenne, pour $\alpha = 0{,}05$, est de $\alpha \cdot (1 - \log_e \alpha) = 0{,}1998$, représentant une réduction de 80 pour-cent par rapport à l'énumération complète; le gain moyen apparaît d'autant plus grand que le seuil de signification est sévère.

Même en pigeant les combinaisons au hasard parmi les T combinaisons possibles, le principe du défoncement de la zone critique s'avère très avantageux. Serait-il possible d'en augmenter l'efficacité en orientant la pige vers des combinaisons défavorables ? On pourrait par exemple faire l'énumération en échantillonnant d'abord les $m = \min(n_1, n_2)$ données les plus faibles, puis les sous-ensembles voisins, ainsi de suite.

Cette procédure de *défoncement optimal* de la zone critique est réalisée suivant deux opérations majeures : le tri des données en ordre décroissant, et l'énumération des combinaisons par un procédé récursif fractionné. Soit l'ensemble trié des $n_1 + n_2$ données, $\{x_1 \geq x_2 \geq \ldots \geq x_{n1+n2}\}$, et $m = \min(n_1, n_2)$. La première somme extrême est $S_m(m) = x_1+x_2+\ldots+x_m$; c'est une somme de niveau $r = m$, et $S_m(m) \geq S_1$. Il faut ensuite trouver toutes les sommes des niveaux $r = m+1$ à $r = n$: pour ce faire, on obtient $S_m(r) = x_r + S_{m-1}\{x_1 \ldots x_{r-1}\}$, où l'expression $S_{m-1}\{x_1 \ldots x_{r-1}\}$ désigne les $C(r-1, m-1)$ sommes de $m-1$ éléments parmi les $r-1$ premiers éléments, l'énumération à cet étage r obéissant au même algorithme présenté plus haut pour l'ensemble des données.

L'amélioration du principe de défoncement grâce à un ordre énumératif optimal vise à hâter l'accumulation de combinaisons à valeurs extrêmes et devrait se montrer utile dans le cas de données à différences non significatives. Les autres principes d'économie, proposés plus bas, touchent l'efficacité globale de l'énumération et devraient se montrer avantageux pour tous les ensembles de données, que leurs différences D_o soient significatives ou non.

Le rebroussement précoce. L'algorithme récursif d'énumération et de sommation procède par étagement récursif, comme son nom l'indique. Soit le groupe-cible avec $m = n_1$ données et la somme S_1. Si toutes les $n_1 + n_2$ données observées sont ordonnées dans une permutation descendante (e.g. $x_1 \geq x_2 \geq \ldots \geq x_{n1+n2}$), on peut anticiper la grandeur des valeurs subséquentes de la somme récursive à un niveau donné et par conséquent ne les énumérer qu'implicitement.

Considérons la somme récursive au niveau final m, S_m. On sait que $S_m = S_{m-1}+x_j$, $j = p_m$. Si $S_m \geq S_1$, il faut comptabiliser cette somme dans la zone critique (en augmentant f_+ ou f_0),

---

d'égalité reste floue. Signalons enfin que $f_0 \geq 1$, puisque, au moins pour le cas de la combinaison observée, $D_p = D_o$.

puis progresser à la S_m suivante, soit $S_{m-1}+x_{j+1}$. Si au contraire $S_m < S_1$, alors puisque les x_i sont décroissants, il est certain que les sommes S_m subséquentes seront elles aussi plus petites que S_1, et l'on peut *rebrousser* à l'étage récursif m–1, avant d'avoir parcouru complètement l'étage m.

Le but de l'énumération étant d'accumuler des valeurs dans la zone critique, la démarche de rebroussement n'aura d'utilité que si elle est suivie de sommes récursives potentiellement critiques (c.-à-d. $S_m \geq S_1$). Or, après avoir trouvé une somme non critique ($S_m < S_1$) et au moment de rebrousser des étages m à m–1, si les indices antérieurs sont compacts (ou consécutifs, c.-à-d. $p_m = p_{m-1}+1 = \ldots = p_i + m-i$) ou bien semi compacts (c.-à-d. $p_m - p_i \leq m-i+1$), alors leurs progressions récursives engendreront des sous-ensembles à valeurs S_m égales ou inférieures au sous-ensemble actuel et elles peuvent être escamotées. On rebroussera donc jusqu'au niveau i pour lequel le sous-ensemble d'indices n'est plus semi compact ; si $i = 0$, l'énumération est terminée. Nous désignons cette généralisation du rebroussement par l'expression « rebroussement en cascade ». Son principe peut s'énoncer comme suit.

> *Principe d'économie 2* :
>
> Posons que les données sont ordonnées (par valeurs décroissantes ou croissantes). Au moment d'obtenir une somme récursive non critique, on peut faire rebrousser l'algorithme à un étage récursif antérieur puisque les sommes suivantes du même étage seront forcément non critiques. L'étage récursif au delà duquel on peut revenir est celui pour lequel le sous-ensemble d'indices positionnels de la somme est semi compact.

Opérationnellement, l'étage-cible de rebroussement correspond à la valeur i pour laquelle ($p_m - p_{i+1}$) $\leq m - i$ et ($p_m - p_i$) $> m - i + 1$.

Le bénéfice du rebroussement en cascade vient de ce qu'il nous permet de seulement survoler les combinaisons non critiques, sans les énumérer explicitement. Après avoir adopté le principe du défoncement de la zone critique comme critère d'arrêt de l'énumération, le principe du rebroussement vient accélérer l'entassement des combinaisons critiques en contournant la majorité des autres combinaisons. Par ailleurs, il est difficile de présenter une analyse quantitative de ce bénéfice ; nous le mesurerons plutôt globalement, lors d'une expérimentation indicative présentée plus loin.

Le plafonnement des indices. Outre le rebroussement précoce, une autre manière de hâter l'achèvement de l'énumération consiste à ne pas l'engager à parcourir un étage récursif s'il est d'avance certain que les sommes encourues seront non critiques. Une fois les données triées en ordre de valeurs décroissantes (ou croissantes), la somme récursive présentera des valeurs plus faibles (ou fortes) si ses indices sont plus élevés; pour un certain sous-ensemble des m indices, il est certain que S_m sera plus faible (plus forte) que S_1, c'est-à-dire non critique. Il est donc superflu de faire encore progresser les indices, d'où le principe du plafonnement. Nous examinerons deux sortes de plafonnement: un plafonnement prévisionnel (ou pré-plafonnement), avec deux variantes, et un post-plafonnement.

- *Pré-plafonnement de l'indice p_1: variante 1*. Soit P, un ensemble des m indices dans la permutation descendante des $n_1 + n_2$ données, et S(P), la somme correspondante; P_o est l'arrangement initial des données du groupe 1, et $S(P_o) = S_1$. Il s'agit de trouver les ensembles P^+ tels que $S(P^+) \geq S_1$: comment construire P^+ ? Posons P = { $p_1, p_2, \ldots, p_m$ } et P_o = { $r_1, r_2, \ldots, r_m$ }, indiquant les positions (les plus élevées en cas d'égalité) dans la permutation descendante globale des x_i. Soit $u = \min(x_{1i}) = x(r_m)$, la plus petite valeur du groupe 1; alors r_m détermine un plafond pour p_1, ce plafond étant $p_1^* = r_m - m+1$. Si de plus P n'est pas compact (selon $r_m - r_1 \geq m$), alors $p_1^* = r_m - m$ est un plafond encore plus efficace. Ce plafond se substitue au plafond récursif normal de $n_2 + 1$, ou $n - m + 1$.

Pour fixer ce plafond de p_1, on doit trouver r_m et r_1. Pour cela, il convient d'obtenir d'abord $u = \min(x_{1i})$ et $v = \max(x_{1i})$, puis r_m = max rang(u) et r_1 = min rang(v), les rangs étant trouvés dans l'ensemble des $n_1 + n_2$ données.

- *Pré-plafonnement de l'indice p_1: variante 2*. Soit SS_j, la somme de m éléments successifs dans la permutation descendante des x_i à partir de la position j: par exemple, $SS_3 = x_3 + x_4 + \ldots + x_{m+2}$. On sait par exemple que $SS_1 \geq S_1$ et $SS_{n-m+1} \leq S_1$; $j = n - m + 1$ est le plafond récursif normal pour p_1. Or il existe vraisemblablement une valeur $j^* < n - m+1$ telle que $SS_{j^*} \geq S_1$ et $SS_{j^*+1} < S_1$: alors $p_1^* = j^*$ constitue un nouveau plafond pour p_1.

Pour déterminer l'économie énumérative attachée au principe du pré-plafonnement, il suffit de décomposer le coût de base C(n, m) en parties subalternes de p_1, soit C(n–1, m–1) + C(n–2, m–2) + … + C(m–1, m–1). Soit le plafond p_1^*; le développement de C(n, m) arrête à C($n - p_1^*$, m–1) inclusivement, d'où le coût après plafonnement est de C(n, m) – C($n - p_1^*$, m). Par exemple, avec $n_1 = n_2 = 8$, C(16,8) = 12870 et le plafond récursif de p_1 est normalement de $n - m+1 = 9$. Un recul de 1 (c.-à-d. $p_1^* = 8$) économise une seule combinaison, un recul de 2 en économise 9, un recul de 6 en économise 10 %, et un recul total (jusqu'à $p_1^* = 1$) n'en garde que 50 %.

La variante 1 du pré-plafonnement se prête à l'analyse. Admettons qu'il n'y a pas de différence réelle entre les groupes et que nous observons deux ensembles de symboles, n_1 symboles « A » (groupe 1) et n_2 « B » (groupe 2), mêlés au hasard. Il y aura recul de 0 position si la

séquence ordonnée débute par un A, et en général de r positions si la séquence débute par r symboles B ; cet événement a pour fréquence relative $p(r) = C(n–1–r, n_1–1) / C(n, n_1)$. Puis le recul sera ajusté, c.-à-d. augmenté de 1, si l'ensemble {A} n'est ni compact ni semi compact dans l'ensemble ordonné. Or, l'ensemble {A} est compact d'une seule façon et semi compact (c.-à-d. intercalé d'un B) de $n_1–1$ façons, la probabilité d'être compact ou semi compact étant donc $p_c(r) = n_1/C(n–1–r,n_1–1)$.

Les expressions ci-dessus permettent d'estimer le recul effectif sous la gouverne de la variante 1 du pré-plafonnement, *en supposant H_0 vraie*. Ainsi, l'espérance de x, le recul effectif, est donnée par $E(x) = \Sigma\ p(r)[x + 1–p_c(r)]$, $r = 0..n_2$. Pour nos conditions $n_1 = n_2 = 8$, nous obtenons $E(x) = 1{,}884$ ($\sigma_x = 1{,}149$), une valeur bien modeste. Les calculs montrent que $E(x)$ varie très peu, s'inscrivant entre 1,8 et 2,0 depuis $n_1 = n_2 = 6$ jusqu'au delà de $n_1 = n_2 = 25$.

Quant à la variante 2, elle dépend de S_1, la somme dans le groupe 1. Sous l'hypothèse nulle de non différence entre les groupes, S_1 fluctue selon $n_1{\cdot}\mu \pm \sigma\sqrt{[n_1n_2/(n–1)]}$. Par exemple, pour $n_1 = n_2 = 8$, cette bande aléatoire est $8\mu \pm 2{,}066\sigma$. La procédure de pré-plafonnement consiste à comparer cette somme S_1 aux sommes basées sur n_1 statistiques d'ordre consécutives et extrêmes, issues des n données. Les probabilités impliquées ici dépendent de la loi de distribution des x, de leurs sommes et de leurs statistiques d'ordre. On peut imaginer cependant que la variabilité réduite de S_1 (du fait que c'est une somme et qu'elle est puisée sans remise dans n données) la confine vers les valeurs centrales de la distribution des sommes ; une hypothèse naïve (et uniforme) la placerait au centre, laissant $½n_2$ sommes consécutives plus extrêmes de part et d'autre et entraînant un recul d'environ n_2.

Le comportement réel de ces principes dépend essentiellement des données qui sont traitées et il doit être évalué par une expérimentation. D'ores et déjà, la variante 2 paraît plus prometteuse que la variante 1.

- *Post-plafonnement*. Les plafonds d'indices peuvent aussi être redéfinis en cours d'énumération. Ainsi, si la somme récursive (de niveau m) S_m est non critique et si les indices successeurs sont tous plus élevés, l'énumération peut être immédiatement stoppée ; cette condition correspond, on le sait, au critère d'ensemble semi compact défini plus haut relativement au principe de rebroussement. Au lieu de l'ensemble complet des m indices, on peut considérer le sous-ensemble des m–1 premiers indices $\{ p_1, p_2,\ldots, p_{m–1} \}$: si ce sous-ensemble est semi compact, tout ensemble successeur compact impliquant p_m ou ses successeurs engendrera aussi une somme non critique, ce qui détermine un plafond pour p_1.

Ce post-plafonnement inclut *de facto* un rebroussement et il s'apparente aussi au rebroussement en cascade. Opérationnellement, à chaque fois que $S_m < S_1$ (c.-à-d. S_m est non critique), il s'agit de vérifier le critère de quasi compacité, selon $p_{m–1} – p_1 \le m – 1$: le cas échéant, le nouveau plafond pour p_1 est de $p1^* = p_m – m$. Le coût courant, après chaque post-plafonnement, s'évalue de la même façon que le pré-plafonnement. La détermination du coût effectif est toutefois plus difficile, en raison du caractère répétitif et aléatoire de ces opérations. Ici encore, seule une expérimentation Monte Carlo permettra d'apprécier l'impact réel de cette stratégie.

Le principe du plafonnement des indices de la somme récursive admet la formulation générale suivante:

Principe d'économie 3:

> L'obtention des sommes successives dans l'énumération se fait par la progression régulière des indices positionnels constituant ces sommes. Si les données destinées à composer la somme récursive sont replacées en ordre de valeurs décroissantes (ou croissantes), alors la progression des indices s'accompagnera généralement d'une diminution (augmentation) de la somme récursive. Comme seules les valeurs critiques (soit $S_p \ge S_1$) de cette somme sont pertinentes, on peut fixer par divers moyens un plafond optimal à la progression des indices, au delà duquel l'énumération ne produirait certainement que des sommes non critiques.

Les divers principes mis de l'avant, avec leurs variantes, constituent des pièces dont il reste à trouver la mosaïque la plus avantageuse. L'expérimentation Monte Carlo, que nous abordons maintenant, nous apportera de l'information permettant de juger les mérites comparés et conjoints de ces différentes stratégies.

Évaluation empirique

L'évaluation mathématique globale des principes d'économie proposés est très complexe car, au contraire de l'algorithme de base qui énumère toutes les combinaisons, ces principes recèlent des éléments stochastiques et, dans le meilleur cas, on ne pourrait les évaluer qu'en moyenne. Nous ressortirons donc à une évaluation de l'efficacité temporelle des algorithmes, par la méthode Monte Carlo. Loin de viser une classification rigoureuse des principes d'économie, notre but est seulement d'indiquer l'ordre de grandeur des performances des différents algorithmes et comment ceux-ci réagissent à une variation des conditions.

Le plan d'attaque de l'évaluation Monte Carlo incluait une série de fichiers de données, codés comme suit: « SxNab », de paramètres « x », « a » et « b ». Chaque fichier contient 25 échantillons, chacun à deux groupes de données normales; « a » est la taille du groupe 1, « b » celle du groupe 2. Le paramètre « x » dénote l'écart paramétrique entre les

Tableau 2: Liste des algorithmes

Algorithme	Description sommaire
Base	Énumération complète et différence de deux moyennes
Réc	Énumération complète et sommation récursive dans le groupe 1, avec différences pré-calculées
D	+ arrêt sur défoncement de la zone critique
Dop	+ arrêt sur défoncement avec énumération optimale
PlA	+ pré-plafonnement en variante 1
PlB	+ pré-plafonnement en variante 2
PlC	+ post-plafonnement
Reb	+ rebroussement précoce
DopReb	+ défoncement optimal avec rebroussement précoce

données des groupes 1 et 2, en unités d'écart-type. Les séries comprenaient les paramètres « a:b » suivants: 8:8, 6:10, 12:12, 9:15.[4] Les écarts « x » quant à eux étaient de 0, 1 et 2. Enfin, le programme de production forçait la présence de 20 échantillons à différence positive non significative et de 5 à différence positive significative, d'après le *t* bilatéral. Ainsi le fichier S1N610 comprenait 25 ensembles de 2 groupes (5 significativement différents, 20 non significatifs) de 6 et 10 données respectivement, chaque donnée du groupe 1 étant par exemple $x_1 = z$, et du groupe 2, $x_2 = z' - 1$, z et z' étant des variables aléatoires normales (σ =1). Du côté des algorithmes comparés, la liste incluait ceux énumérés au Tableau 2.

Rappelons que les différents algorithmes sont d'une certaine façon cumulatifs. Les algorithmes de défoncement (simple ou avec énumération optimale) utilisent la sommation récursive (Réc). Quant aux algorithmes PlA, PlB, PlC et Reb, ils utilisent tous la sommation récursive et le défoncement simple. Enfin, soulignons que les algorithmes Dop, PlA, PlB, PlC, Reb et DopReb requièrent que les données soient placées en ordre (décroissant) ; les temps d'exécution comprennent donc la mise en ordre des données.[5]

Les graphiques de la Figure 5, à la page suivante, suffisent à indiquer l'allure générale des performances. Pour des données contenant une différence éventuellement significative, les seuls principes de défoncement (D, Dop) détériorent plutôt la performance puisque, dans ces cas, la zone critique n'est pas défoncée et toutes les C(n, m) combinaisons sont énumérées ; l'avantage du défoncement advient pour les cas à différence non significative, le défoncement avec énumération optimale devançant quelque peu le défoncement simple.

La Figure 5, sur échelle logarithmique, montre à l'évidence la supériorité du principe de rebroussement précoce, que ce soit pour des données à différence significative ou non. L'algorithme Reb, qui exploite aussi le principe de défoncement, escamote l'énumération des sommes récursives avérées non critiques; c'est comme si l'algorithme Reb faisait l'*énumération implicite* de ces sommes, tandis que les algorithmes à plafonnement (PlA, PlB, PlC) préviennent l'énumération des sommes qu'on a prédites non critiques. Remarquer enfin que l'algorithme DopReb, qui combine le rebroussement précoce et l'énumération optimale, est presque aussi performant que le rebroussement simple. En fait, dans la plupart des cas, le nombre de combinaisons (ou sommes) étudiées par DopReb égale à peu près celui de l'algorithme Reb mais la durée par combinaison pour DopReb est de 30 à 75 % plus grande que pour Reb. L'augmentation de coût entraînée par l'énumération optimale apparaît aussi quand on compare les algorithmes D et Dop; la durée par combinaison pour Dop est de 5 à 6 % supérieure à celle pour D, cet inconvénient se manifestant directement dans le cas de comparaisons significatives.

Quant aux quelques paramètres de séries étudiés, à savoir l'écart « x » (= 0σ, 1σ ou 2σ) entre les données des deux groupes et les tailles de ces groupes, leurs effets particuliers sur la performance des algorithmes ne sont pas

[4] Rappelons que l'étude initiale a été réalisée en 1993, sur un ordinateur moins rapide. Des informations de mise à jour sont fournies en épilogue de l'article.

[5] L'algorithme de tri utilisé est celui de la *sélection simple* ; l'*insertion simple* (ou avec sentinelle) aurait aussi fait l'affaire (voir Knuth, 1969 ou Laurencelle, 1977). Une courte étude d'efficacité montre que, pour des séries de taille ≤ 50, les algorithmes simples restent préférables aux algorithmes plus sophistiqués, tels *Quicksort*, *Heapsort* ou *Shellsort*.

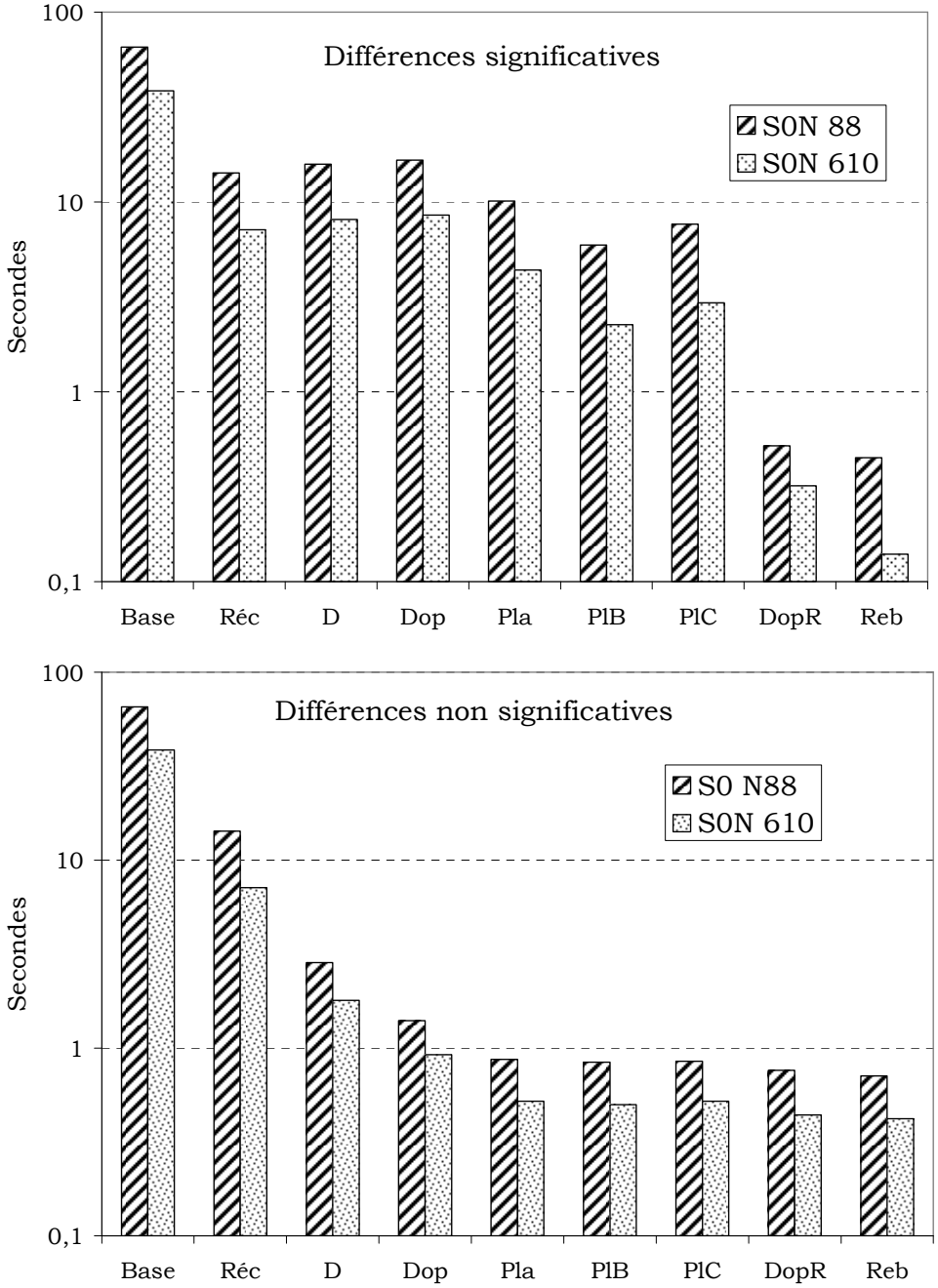

Figure 5. Durées d'exécution de différents algorithmes de comparaison de deux moyennes par combinatoire exhaustive donnant lieu à une différence significative (en haut) ou non significative (en bas)

clairs. Pour les petits échantillons (soit a = 8:b = 8 et a = 6:b = 10), la durée d'exécution augmente avec l'écart « x » dans les séries à différence significative et elle décroît dans les séries à différence non significative ; de même, le temps par combinaison est plus fort pour les premières que pour les secondes. Les ratios de durées d'exécution, pour deux algorithmes comparés, varient d'une taille de séries à l'autre, ce qui rend la projection de temps hasardeuse. Dans tous les cas toutefois, l'algorithme Reb remporte la palme. Par exemple, pour la série S0N915, les durées associées à Reb sont 58,046 et 89,847 s, comparativement à PlA avec 780,536 et 92,260, à PlB avec 635,770 et 89,506 et à PlC avec 692,825 et 92,638.

Nous rapportons ici les résultats de quelques tests seulement, mais nous avons également mis à l'essai diverses combinaisons des principes d'économie proposés. Ces combinaisons réunissaient par exemple le rebroussement et l'un ou l'autre des pré-plafonnements. Toutefois, il a fallu constater que l'efficacité même du rebroussement en cascade rend toute amélioration, non pas impossible, mais difficile.

Cette difficulté s'est manifestée clairement dans l'algorithme combiné (désigné DopReb), réunissant défoncement avec énumération optimale et rebroussement. Compétiteur sérieux de l'algorithme Reb, l'algorithme DopReb est parvenu à quelques reprises à réduire l'énumération de quelques unités, mais sa durée d'exécution s'est montrée légèrement et systématiquement plus longue, l'effet indubitable du fractionnement de la récursion. Le mérite efficace de l'algorithme Reb reste probablement sa grande simplicité.

Conclusion et épilogue

L'algorithme Reb, mettant en œuvre le principe de rebroussement précoce, s'avère donc le plus efficace pour la comparaison de deux moyennes indépendantes par combinatoire exhaustive. L'annexe présente un programme complet en QBasic, programme clef-en-main, qui accommode toutes conditions spécifiées par l'utilisateur. Son temps d'exécution dépendra des tailles d'échantillons fournies et de la machine employée. Sur notre compatible-IBM à processeur Intel Pentium à 2,8 GHz, pour traiter un échantillon constitué de deux groupes de 14 données (a=14:b=14), la durée effective pour l'algorithme de base (Base) est d'environ 29m58s; pour l'algorithme simplifié avec calcul récursif des sommes (Réc), la durée projetée est de 5m40s. Exécutant l'algorithme Reb de rebroussement précoce avec les mêmes fichiers de données, les durées mesurées variaient de 0m0s60cs à 0m6s64cs pour les comparaisons significatives et de 0m11s26cs à 0m11s37cs pour les non significatives, des durées bien raisonnables.

Le lecteur aura compris que le facteur déterminant pour le coût temporel de l'exécution informatique se rapporte au nombre total de permutations à considérer, plutôt qu'au seul nombre total de données *n.* Ainsi, pour nos $n = 28$ données réparties en deux groupes, une répartition 20:8 génère environ 13 fois moins de combinaisons que la répartition 14:14 considérée. Avec 20:8, le temps requis pour l'algorithme de base est passé de 29m58s à 2m11s, pour Réc, de 5m40s à 20s et pratiquement à 0s pour notre algorithme Reb à rebroussement précoce.

Des ordinateurs plus performants, comme il s'en annonce, permettront sans doute d'élever quelque peu le plafond des tailles de groupes accessibles. Il ne faut pas rêver cependant, en raison de la progression quasi exponentielle du coût en fonction de n : ainsi, pour $n_1 = n_2 = 20$, le nombre de combinaisons à considérer est de 137.846.528.820, et même un cycle d'exécution informatique rapide, multiplié par une petite fraction de ce nombre, restera longtemps un défi. Le recours à la combinatoire approximative, voire aux approximations basées sur la loi normale, devra encore être envisagé.

Références

Bradley, J.V. (1968). *Distribution-free statistical tests*. Englewood Cliffs (NJ), Prentice-Hall.

Edgington, E.S. (1969). Approximate randomization tests. *Journal of psychology*, *72*, 143-149.

Edgington, E.S. (1980). *Randomization tests*. New York, Marcel Dekker.

Hope, A.C.A. (1968). A simplified Monte Carlo significance test procedure. *Journal of the Royal Statistical Society B*, *30*, 582-598.

Knuth, D. E. (1969). *The art of computer programming. Volume 2 : Seminumerical algorithms.* Reading : Addison-Wesley.

Laurencelle, L. (1977). Algorithmes de tri interne sur ordinateur. *Lettres Statistiques*, *3*, 40 p.

Laurencelle, L. (2001). *Hasard, nombres aléatoires et méthode Monte Carlo*. Québec : Presses de l'Université du Québec.

Laurencelle, L. (2012). Le nombre de permutations dans les tests permutationnels. *Tutorials in Quantitatives Methods for Psychology*, *8*, 1-10.

Lehmann, E.L. (1975). *Nonparametrics: Statistical methods based on ranks*. San Francisco, Holden-Day.

Marriott, F.H.C. (1979). Barnard's Monte Carlo tests: how many simulations? *Applied Statistics*, *28*, 75-77.

Siegel, S., Castellan, N. J. *Nonparametric statistics fort he behavioral sciences* (2e edition). New York : McGraw-Hill.

Appendix follows

Appendix: programme de langage BASIC

```
'
' Programme BASIC de comparaison de deux moyennes indépendantes
' par combinatoire exhaustive avec rebroussement précoce
'
' Pierre Ferland & Louis Laurencelle 1993, rév. 2011
'
' *** Paramétrisation du problème
'
        DEFDBL A-Z
        INPUT "n1,n2 "; n1, n2
        n = n1 + n2
        IF n1 <= n2 THEN M = n1 ELSE M = n2
        DIM X(n), D(n), P(M), PM(M)
        S1 = 0: S2 = 0

        PRINT "Inscrire les données du groupe 1 :"
        FOR i = 1 TO n1: INPUT X(i): S1 = S1 + X(i): NEXT
        PRINT "Inscrire les données du groupe 1 :"
        FOR i = n1 + 1 TO n: INPUT X(i): S2 = S2 + X(i): NEXT
'
' *** Calculs préliminaires
'
        Dif = S1 / n1 - S2 / n2
        PRINT "La différence des moyennes est "; Dif
        IF Dif = 0 THEN PRINT "Salut bien!": END
        PRINT "Inscrire le seuil (unilatéral) du test voulu ou 0 pour aucun test :"
10      INPUT Alpha
        IF Alpha >= .5 THEN PRINT " *** attention : de 0 a 0.5 seulement...": GOTO 10
        IF Alpha = 0 THEN PRINT "Salut bien!": END
'
' *** Préparation de l'énumération
' *** Détermination de l'ordre croissant ou décroissant du tri
' ***  puis tri par sélection simple (peut être modifié au choix)
' *** Détermination des plafonds d'indices (PM)
' *** Calcul des différences successives (D) et pré-soustraction de S1
'
        t = 1: FOR i = 0 TO M - 1: t = t * (n - i) / (M - i): NEXT
        TC = Alpha * t
        PRINT t; " arrangements différents"
        PRINT INT(TC * 10) / 10; " arrangements critiques"
        IF n1 <= n2 THEN S = -S1 ELSE S = -S2: Dif = -Dif
        IF Dif > 0 THEN W = 1 ELSE W = -1
        FOR i = 1 TO n - 1: C = X(i): k = i
          FOR j = i + 1 TO n: IF W * (X(j) - C) > 0 THEN C = X(j): k = j
          NEXT: X(k) = X(i): X(i) = C
        NEXT
        FOR i = 1 TO M: PM(i) = n - M + i: NEXT
        FOR i = 2 TO n: D(i) = X(i) - X(i - 1): NEXT
'
```

```
' *** Énumération et calcul récursif de la somme, avec rebroussement précoce
'
        XT = 0: U = 0: M1 = M + 1
        i = 1: P(0) = -9: P(1) = 1: S = S + X(1): GOTO 40
20      IF W * S < 0 THEN 55
        XT = XT + 1: IF XT > TC THEN U = 1: GOTO 60
30      IF P(i) = PM(i) THEN 50
        j = P(i) + 1: P(i) = j: S = S + D(j)
40      IF i = M THEN 20
        i = i + 1: P(i) = P(i - 1) + 1: S = S + X(P(i)): GOTO 40
50      S = S - X(P(i)): i = i - 1: IF i > 0 GOTO 30
        GOTO 60
55      S = S - X(P(i)): i = i - 1: IF P(M) - P(i) <= M1 - i GOTO 55
        IF i > 0 GOTO 30
'
' *** Conclusion du test ( U = 1 lorsque la zone critique est défoncée )
'
60      t2 = TIMER
        IF U = 0 THEN
        PRINT USING "Difference significative (p = #.######)"; XT / t
        ELSE
        PRINT USING "Difference non significative (p > #.####)"; Alpha
        END IF
       END
```