Appariement de matrices de dissimilarités

Daniel Defays

Université de Liège
ddefays@ulg.ac.be

Résumé. Dans cet article, nous comparons différentes méthodes qui permettent de trouver dans un ensemble fini S, sur lequel on a défini une dissimilarité, un sous-ensemble d'éléments qui s'apparient au mieux aux éléments d'un ensemble cible C également muni d'une dissimilarité. La qualité de l'appariement est mesurée par la distance observée entre les matrices de dissimilarités définies sur les sous-ensembles de S et sur C. Différents algorithmes sont présentés et les performances sont commentées.

Mots clefs. Distances – Espaces métriques - Appariement – Isométrie - Reconnaissance de formes

Summary. In this paper, we compare different methods to map in a optimal way a subset of a finite set S - on which a dissimilarity is defined – on a target set C, on which another dissimilarity is defined. The quality of the mapping is assessed by comparing the dissimilarities of the elements of the target C with the corresponding dissimilarities in S. The closer they are, the better the fit is. Different algorithms are presented and performances are compared.

Key words. Distances - Metric spaces - Mapping – Isometric mapping - Pattern recognition

1. Introduction

Pourrait-on associer des photos à des phrases musicales et ainsi interpréter une œuvre avec ces photos? Pourrait-on dans la même logique associer des images à des parfums, des œuvres musicales à des vins ? Ces questions sont à la base d'un projet de recherche au sein duquel les appariements de structures traités dans cet article jouent un rôle essentiel. Le problème étudié est fort général, en fait : comment reconnaître dans un ensemble donné, caractérisé par une mesure de dissimilarité, des formes particulières définies par ailleurs.

Différents types de réponse ont été donnés jusqu'à présent dans le domaine qui nous intéresse. Les associations sons, couleurs ont été appréhendées dans de nombreuses œuvres artistiques par des peintres abstraits comme Vassily Kandinsky, Paul Klee, des musiciens comme Modeste Moussorgski, Nikolaï Rimsky-Korsakov, des cinéastes comme Mary Ellen Bute (considérée comme une des pionnières de la musique visuelle), ou Norman McLaren (Bos-

seur, 2006). Des logiciels comme Media Player associent des animations graphiques à des airs de musique et des équipes multidisciplinaires explorent avec des dispositifs électroniques les relations qui existent entre les sons, les couleurs, la lumière, le mouvement et les formes. Jack Ox et David Britton, par exemple, ont conçu un dispositif appelé « color organ » qui permet de traduire des compositions musicales en images de synthèse représentant des paysages qui se déforment en suivant les inflexions de la musique (Ox et Britton, 2000).

Des travaux qui utilisent des techniques de fouille de données (Widmer et al, 2003) ont permis de représenter graphiquement dans des espaces à deux dimensions des interprétations musicales en utilisant les modulations du tempo et du volume appliquées par le musicien pour donner du relief à son interprétation. Visuellement, l'interprétation est caractérisée par un vers qui progresse dans un espace à deux dimensions.

Les approches mathématiques se sont concentrées soit sur l'extraction, à partir d'une seule matrice de dissimilarités, de formes définies a priori – points sur une droite, points dans un espace multidimensionnel de dimension faible, hiérarchies, pyramides (Diday et al, 2008) - soit sur des projections isométriques de points d'une surface sur un plan … peu ou pas à notre connaissance sur l'appariement de formes définies par un nombre fini d'éléments dans des ensembles différents. Par contre, des algorithmes fort généraux de type Metropolis-Hastings, existent et permettent de trouver des solutions approchées lorsque la taille des ensembles ne permet pas de faire mieux. Remarquons également que l'extraction de formes définies a priori permet de simplifier le problème d'appariement traité dans cet article. Il ramène en effet ce problème à un appariement de formes simples et non de matrices quelconques mais il postule un modèle de données qui ne se retrouve pas toujours dans les configurations étudiées. Techniquement, le problème se ramène à trouver une transformation isométrique (en fait aussi isométrique que possible) d'un espace dans un autre.

Cet article, après avoir présenté le contexte empirique dans le lequel s'inscrit la démarche, propose et compare différents algorithmes pour apparier des structures à partir de matrices de dissimilarités définies dans des espaces différents.

2. Un exemple de champ d'application

Plus concrètement, le projet qui a motivé la recherche d'algorithmes d'appariement de structures se formule comme suit (Defays, 2017). Un morceau de musique peut se décomposer en segments et une mesure de dissimilarité entre ces segments peut être calculée – par exemple en utilisant des analyses spectrales et des « Mel-Frequency Cepstral Coefficients ». Le morceau est ainsi représenté par une matrice de dissimilarités dont la dimension est égale au nombre de segments considérés. Elle définit une forme, une trajectoire quelconque – la ligne brisée qui relie les représentations des segments dans un espace multidimensionnel. L'ensemble de segments sera noté C dans la suite de cet article et appelé l'ensemble cible. La matrice aura typiquement dans le projet de recherche qui a motivé le travail présenté une dimension de l'ordre de 5 à 50. De manière identique, un ensemble d'images, appelé l'ensemble source et noté S pourra aussi être structuré au moyen d'une distance ou plus faiblement d'une dissimilarité. L'utilisation de mots visuels et d'analyses des correspondances comme proposé par Pham, Morin, Gros et Le (2009) permet ce type d'analyse. L'ensemble source pourra cependant typiquement comporter de 50 à plusieurs centaines d'éléments. Le problème est donc de trouver dans cette librairie d'images le sous-ensemble ordonné

d'éléments – que nous appelons une bandelette musicale - qui s'ajuste au mieux aux segments musicaux. L'ajustement sera considéré comme bon lorsque les dissimilarités entre les images sélectionnées sont proches des dissimilarités entre les segments musicaux correspondants. A deux segments musicaux proches doivent donc correspondre deux images similaires.

3. Terminologie et formulation mathématique

Soit C un ensemble $\{c_1, c_2, c_3 \ \ldots \ c_N\}$ d'effectif N sur lequel on a défini une dissimilarité d. A chaque paire d'éléments $\{c_i, c_j\}$ est donc associée une valeur $d(c_i, c_j)$ (= $d(c_j, c_i)$) qui mesure leur dissimilarité. De même, un ensemble S, dont l'effectif |S| est supérieur à N est muni d'une dissimilarité d' qui chiffre la « distance » entre ses paires d'éléments.

Si f est une application injective de C sur S, l'image de C – notée f (C) - est un sous-ensemble de S d'effectif N. La « distance » entre ce sous-ensemble et C sera mesurée par

$$\Delta\,(C, f(C)) = \sum_{i,j}^{N} |d\,(c_i, c_j) - \lambda\, d'(f(c_i), f(c_j))\,|$$

avec $\lambda = \frac{\sum_{i,j}^{N}(d(c_i,c_j)}{\sum_{i,j}^{N}(d'(f(c_i),f(c_j))}$

λ représente un facteur de correction d'échelle car les dissimilarités d et d' n'utilisent pas a priori la même unité de mesure. Il impose simplement que les dissimilarités moyennes soient les mêmes dans C et f(C). Ce facteur a un caractère relativement arbitraire et d'autres choix pourraient être considérés pour rendre les deux métriques comparables (par exemple utiliser des variances et non des moyennes).

La fonction f définit un appariement de S à C et l'élément s = f(c) est dit apparié à c.

Comment trouver l'injection f qui minimise Δ, c'est-à-dire comment trouver un appariement optimal ? Le nombre de possibilités à examiner si l'on procède de manière exhaustive explose rapidement avec la taille des ensembles C et S puisqu'il faut considérer a priori $A_{|S|}^{N}$ configurations différentes dans S. Ainsi, pour reprendre l'exemple donné en introduction, en partant d'un morceau de musique décomposé en 24 segments et d'un ensemble de 155 images, il faudrait considérer 5.63×10^{51} possibilités différentes. La recherche d'algorithmes qui permettent de trouver une solution à ce problème est l'objet de cet article.

4. L'algorithme de Metropolis-Hastings

La recherche de l'injection f est un problème classique de minimisation d'une fonction (la distance Δ) sur un grand ensemble fini – S^N. Une méthode de Monte-Carlo, comme l'algorithme de Metropolis-Hastings, peut être appliquée. Il s'agit de minimiser Δ en parcourant, a priori, tous les sous-ensembles ordonnés possibles de S que nous appelons des trajectoires, entreprise impossible comme nous l'avons signalé à partir d'une recherche exhaustive. Pour pouvoir appliquer l'algorithme de Metropolis-Hastings, l'ensemble S^N des trajectoires possibles dans S doit être muni d'une structure de graphe. Ceci peut se faire de différentes

manières en garantissant la nécessaire connexité. Deux trajectoires $(x_1, x_2, \ldots x_N)$ et $(y_1, y_2, \ldots y_N)$ seront considérées liées si une des deux conditions suivantes est satisfaite :

1. elles différent uniquement par un élément (pour tous les i sauf un, $x_i=y_i$) ; les deux trajectoires sont donc identiques à un point près ;
2. elles sont égales à une permutation de deux éléments près.

Il suffit alors de parcourir le graphe en cherchant à minimiser la distance Δ – en se réservant la possibilité de ne pas prendre le chemin optimum dans certains cas pour éviter autant que faire se peut les minima locaux. L'algorithme a été appliqué en partant d'une trajectoire X^0 tirée en choisissant au hasard N éléments dans S, en calculant la distance Δ^0 à la cible puis en choisissant, de nouveau de manière aléatoire un voisin direct de X^0, noté X^1. Deux cas sont alors à distinguer. Soit X^1 est meilleur que X^0, (c'est-à-dire que cette configuration est plus proche de la cible) et dans ce cas, on remplace X^0 par X^1 ; soit ce n'est pas le cas, et on ne remplace X^0 par X^1 qu'avec une probabilité égale à $e^{-\beta(\Delta^1-\Delta^0)}$ en notant Δ^1 la distance de X^1 à la cible. β est un paramètre qui permet de contrôler la probabilité avec laquelle on s'autorise de faire « marche arrière » pour éviter les optima locaux.

Pour affiner l'algorithme, nous avons appliqué un deuxième niveau de recuit simulé partiellement contrôlé dans l'algorithme de Metropolis-Hastings par la valeur de β. Pour rappel, le recuit simulé permet d'enchaîner une recherche où l'on se permet assez fréquemment de revenir en arrière – pour ce faire, dans notre cas β doit être petit - avec des recherches où ces retours deviennent de plus en plus exceptionnels au fur et à mesure que l'on se rapproche d'une solution. Il s'agit donc d'appliquer itérativement l'algorithme en augmentant progressivement la valeur de β (elle est doublée à chaque vague d'itérations) jusqu'à ce que l'approximation trouvée se stabilise. L'algorithme ainsi construit est appelé APPARIEV0.

L'algorithme de Metropolis-Hastings présente, dans l'application qui nous concerne, les avantages et inconvénients suivants. Il est très facile à mettre en œuvre car la méthode est simple ; il se montre très performant – en qualité de l'approximation et en temps d'exécution - sur les grands ensembles de données car les temps dépendent plus du nombre d'itérations que de la taille de ces ensembles et l'algorithme converge assez rapidement. Cette technique est donc à conseiller lorsque le nombre d'images à apparier et le nombre de segments musicaux sont importants, les seuils dépendant des machines utilisées et du temps disponible (voir les simulations réalisées dans la suite de cet article). Par contre, elle ne permet pas d'obtenir à coup sûr la meilleure solution et de contrôler facilement la qualité de l'ajustement. Si la bandelette musicale se veut une interprétation visuelle du morceau de musique, il faut éviter toutes les « fausses notes ». Une fausse note apparaît lorsqu'un lien est établi entre un élément c et un élément s dont la position relative par rapport aux autres éléments de f(C) est très différente de la position relative de c au sein de C. Ceci nécessite de rendre impossible d'apparier une image à un segment si la distribution de ses dissimilarités aux autres images ne respecte pas la distribution des dissimilarités du segment aux autres segments. Or les fausses notes sont possibles lorsque la seule préoccupation est de minimiser Δ. Le paragraphe qui suit présente deux variantes d'une méthode qui permet de mieux contrôler la qualité générale des bandelettes avec des ensembles de tailles raisonnables (dans le contexte de notre exploration) : des cibles d'une dizaine de segments et quelques centaines de photos.

5. Principes de la nouvelle méthode proposée

La nouvelle méthode proposée repose sur les trois observations suivantes.

S'il existe un appariement parfait (distance Δ entre C et f(C) nulle), dès que deux éléments de S (appelons-les s_1 et s_2) sont appariés à deux éléments de C (c_1 et c_2), le problème d'échelle est résolu : λ vaut $\frac{d(c_1,c_2)}{d(s_1,s_2)}$

Si l'élément s de S doit être apparié à l'élément c de C, et si s_1 et s_2 ont été appariés respectivement à c_1 et à c_2, les dissimilarités « corrigées » λd' entre s et s_1 et entre s et s_2 ne doivent pas être trop différentes, respectivement, des dissimilarités d entre c et c_1 et, c et c_2.

De manière plus générale, si les éléments notés s_1, s_2, s_3 … s_n ont déjà été appariés à c_1, c_2, c_3 … c_n, alors si s (différent de s_1, s_2, s_3 … s_n) est lié à c (différent de c_1, c_2, c_3, … c_n) , les dissimilarités corrigées entre s et les s_j (pour $j \leq n$) doivent être respectivement similaires aux dissimilarités entre c et les c_j (pour $j \leq n$) (pour éviter les « fausses notes »).

Sur base de ces constats, un algorithme récursif simple peut être écrit. Les principes de fonctionnement sont les suivants.

3. tous les couples possibles d'éléments de S sont considérés tour à tour ;
4. pour un couple (s_1, s_2) donné, on analyse tous les appariements pour lesquels s_1 est lié à c_1 et s_2 à c_2 ;
5. pour ces appariements, les dissimilarités sont ajustées de manière à ce que λd'(s_1, s_2) =d(c_1, c_2) ;
6. pour ces appariements, pour chaque i>2, on identifie les éléments de s a priori appariables à c_i en utilisant le constat 2. Il importe cependant de fixer un seuil (noté tol) pour donner un sens à « pas trop différent ». Nous pouvons par exemple imposer que la différence relative entre λd'(s, s_1) et d(c_i, c_1) ($|\lambda$d'(s, s_1) –d (c_i, c_1)|/ d(c_i, c_1)) soit inférieure à tol (idem pour s_2 et c_2). Cette contrainte permet de limiter le nombre de configurations à considérer et de s'assurer que les appariements recherchés ne seront jamais de trop mauvaise qualité. Dans un grand nombre de cas – et la simplification de la recherche vient de là - la contrainte exclura tout type d'appariement qui lierait s_1 à c_1 et s_2 à c_2, car pour certains c_i (i>2) il n'existera pas de correspondants satisfaisants dans S. Le couple (s_1, s_2) pourra alors être ignoré ;
7. si (s_1, s_2) ne peut pas être ignoré suite à la contrainte imposée en 4, on considère tour à tour tous les éléments de s appariables à c_3. Pour chaque possibilité d'appariement d'un s à c_3, on revoit la liste des éléments appariables à c_4, c_5 … en utilisant le troisième constat. Si à la suite de cet examen, il s'avère que pour un c_i (i>3) il n'existe plus d'éléments de S appariables, l'élément s est abandonné et un autre élément s appariable à c_3 est considéré ;
8. si ce n'est pas le cas, les listes d'éléments de S appariables aux c_i (avec i>3) sont mises à jour pour tenir compte du lien (s, c_3) et on procède de manière itérative en considérant un élément s qui puisse être apparié à c_4, cette fois. De nouveau, la liste des éléments appariables est mise à jour pour tenir compte de cette nouvelle contrainte. Si cette mise à jour révèle que pour un c_i (avec i>4) il n'existe plus d'éléments de S appariables, le lien (s, c_4) est abandonné et un autre lien (s, c_4) est considéré ;

9. on procède ainsi jusqu'à obtenir un premier ajustement (s_1, s_2,… s_N) à (c_1, c_2, … c_N). On mesure la qualité de cet appariement en calculant la distance Δ et on poursuit l'itération ;
10. lorsque tous les couples possibles d'éléments de S ont été considérés, on sélectionne l'appariement qui donne la distance minimum Δ.

Cet algorithme simplifié – appelé APPARIEV1 - permet toujours d'obtenir une ou la solution exacte (Δ = 0) - si elle existe - pour des raisons évidentes : en effet, comme mentionné ci-dessus, dans ce cas λ vaut $\frac{d(c_1,c_2)}{d(s_1,s_2)}$, et tous les s_i de la solution seront gardés dans les listes des éléments appariables puisque, en notant a = d(c_1, c_2), b = d'(s_1, s_2), u = d(c_i, c_j), v = d'(s_i, s_j) avec i<j, on aura toujours |u - $\frac{a}{b}$ v| = 0.

De même s'il n'existe pas de solution exacte mais une solution approchée de bonne qualité (à distance Δ <<), l'algorithme permettra de la trouver pour autant qu'elle respecte la contrainte imposée sur la qualité des solutions, à savoir que toutes les dissimilarités corrigées entre les s_i ne soient pas trop différentes des dissimilarités entre les c_i. On notera en effet que

$$\left|u - \frac{a}{b}\ v\right| = \left|u - \ \lambda v + \ \lambda v - \frac{a}{b}\ v\right| \leq |u - \ \lambda v| + v\left|\lambda - \frac{a}{b}\right|$$

Si Δ est petit, $|\lambda - \frac{a}{b}|$ le sera également et par conséquent les s_i de la solution seront gardés dans les listes des éléments appariables, si le seuil tol n'est pas pris trop petit.

6. Test des algorithmes « APPARIEV0 » et « APPARIEV1 »

De manière à pouvoir comparer les performances de cet algorithme avec celles d'une recherche exhaustive et de l'algorithme de Metropolis-Hastings, des configurations cibles et des fichiers sources relativement simples ont été considérés.

Trois types de configurations cibles ont été définies pour les simulations : 5 points alignés ; 5 points situés sur un carré ; 5 points situés sur une « parabole ». Les 3 cibles sont reprises ci-dessous.

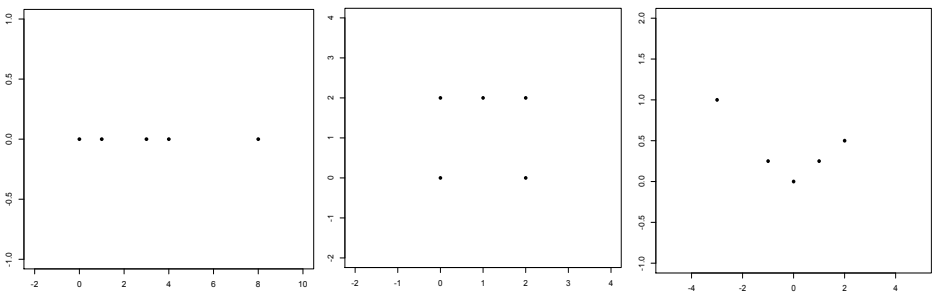

FIG. 1 - *Trois ensembles Cibles différents*

De même, un ensemble de 40 points distribués de manière aléatoire dans le rectangle [1,500] x [1,1000] a été généré. Il constitue l'ensemble source S. La distribution des points est donnée dans la figure 2.

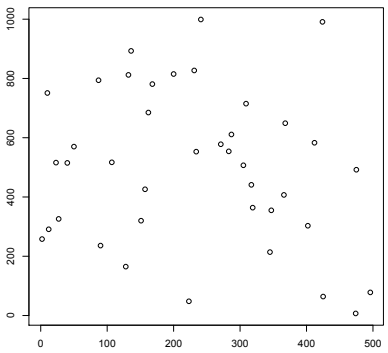

FIG. 2 - *Ensemble Source*

Les ajustements optimaux (ce sont les meilleurs possibles avec les données simulées) sur les 3 cibles sont donnés dans la figure 3 ci-dessous.

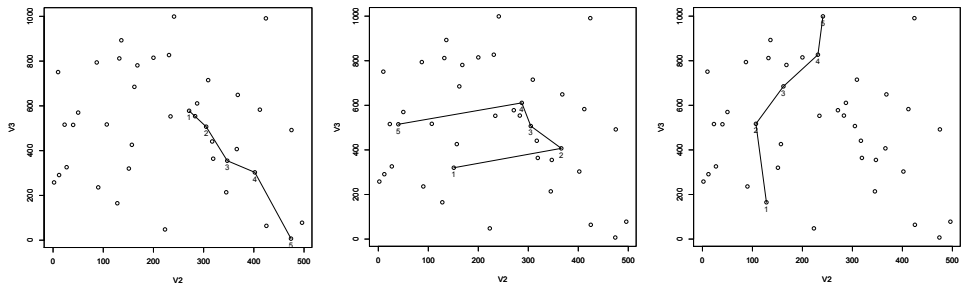

FIG. 3 - *Ajustements optimaux pour les 3 cibles : ligne, carré, parabole*

L'algorithme APPARIEV1 permet de réduire le temps de recherche (par rapport à une recherche exhaustive) par un facteur variant avec le seuil utilisé. Avec une valeur de 0.2 (on tolère donc des distorsions des dissimilarités de 20% maximum), le temps pour trouver la solution optimale est réduit par un facteur de l'ordre de 1500 (le temps nécessaire passe d'environ 30 minutes à 1.2 secondes avec un programme écrit en R utilisant la fonction « permutations » de la librairie gtools sur un Imac avec un processeur 2.9 GHz Intel Core i5). Pour rappel, il y a 78 960 960 configurations à examiner a priori dans cette simulation. APPARIEV0 prend un peu plus de temps mais en restant largement en dessous de la minute.

Le tableau ci-dessous donne les temps nécessaires en secondes pour trouver la solution avec les différentes configurations et différentes valeurs du seuil. Les deux dernières lignes présentent le temps pour la recherche exhaustive et le temps moyen pour une solution approchée avec l'algorithme de Metropolis-Hastings.

Pour ce dernier, dans les simulations réalisées, une première vague d'itérations (50000) a été exécutée avec une valeur 0.1 du paramètre β, puis une nouvelle vague de 50000 itérations a suivi en doublant β et en partant de la dernière configuration obtenue, et ainsi de suite jusqu'à stabilisation du résultat sur deux vagues successives. Pour chacune des 3 cibles, 100 simulations ont été réalisées et les moyennes du temps et de la distance Δ ont été calculées.

Seuil (tol)	Ligne	Carré	Parabole
0.1	0.93	0.90	0.91
0.2	1.1	1.29	1.27
0.3	1.2	2.21	2.1
0.4	1.92	4.73	3.97
0.5	4.12	14.17	8.82
0.6	10.9	55	22.9
0.7	34	194.4	71
0.8	126	478	250
Recherche exhaustive	2209	1876	1884
APPARIEV0	20	20	24

TAB 1 – *Temps en secondes mis par les différents algorithmes pour différentes valeurs de paramètres pour trouver la solution*

La qualité de l'approximation obtenue par l'algorithme de Metropolis-Hastings peut s'apprécier dans le tableau 2 qui compare les distances Δ des configurations optimales à la cible aux distances moyennes obtenues avec cet algorithme.

Algorithme	Ligne	Carré	Parabole
Recherche exhaustive	0.92	1.90	0.73
APPARIEV0	1.26	2.28	1.04

TAB 2 – *Distances à la cible des différentes solutions obtenues par les algorithmes pour trouver la solution en fonction du seuil et du type de cible*

Le choix du seuil est bien entendu critique, comme le confirme le tableau 1, car c'est lui qui contrôle le nombre de configurations examinées. La nature de la cible est aussi importante car elle détermine également le nombre de solutions potentielles à examiner. La réduction du nombre de configurations à analyser ou du temps mis pour obtenir la solution obtenue en appliquant APPARIEV1 est cependant loin d'être suffisante pour pouvoir résoudre des problèmes plus réalistes d'appariement. Ainsi dans le cadre du projet d'appariement d'une séquence d'images sur une œuvre musicale, le temps nécessaire, pour un ensemble source de 155 photos, à apparier sur un morceau de musique décomposé en 10 parties (la cible), s'est chiffré en jours ; Metropolis-Hastings propose une solution approchée après une quinzaine de secondes (avec les mêmes paramètres que dans la simulation).

7. Nouvelle méthode pour traiter des ensembles de données plus importants

APPARIEV2 présenté ci-dessous est une variante de APPARIEV1, plus rapide mais malheureusement, qui ne garantit pas toujours de trouver d'aussi bonnes solutions que la version précédente. Elle se base, comme la version 1, sur les trois premiers constats faits précédemment, mais aussi sur la considération additionnelle suivante : le troisième constat est d'autant plus contraignant que n (le nombre d'éléments déjà appariés) est grand.

Ainsi, lorsque les deux premiers éléments de S ont été choisis (ils fixent – rappelons-le – l'échelle des dissimilarités et tous les couples d'éléments sont considérés tour à tour), la liste des éléments potentiellement appariables au troisième élément de la cible c_3 peut être longue. En effet, il est simplement exigé d'un élément s pour faire partie de cette liste, que deux paires de dissimilarités ($\{\lambda$ d'(s, s_1), d(c_3, c_1)$\}$ et $\{\lambda$ d'(s, s_2), d(c_3, c_2)$\}$) soient similaires.

Au fur et à mesure que l'on ajoute des éléments s_i à apparier aux c_i, le nombre de contraintes pour les éléments qui restent à apparier augmente comme l'illustre la figure 4.

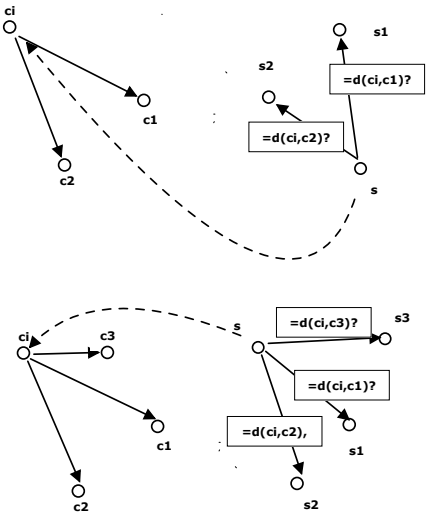

FIG. 4 - *Contraintes imposées à l'appariement de s à c*

Au début, s_1 est apparié à c_1 et s_2 à c_2. Un élément s quelconque peut donc a priori être apparié à c_i si deux contraintes seulement sont satisfaites comme illustré ci-dessus. Ceci laisse encore des degrés de liberté dans le choix de s. Lorsque, à l'étape suivante, un élément s_3 est apparié à c_3, une contrainte supplémentaire doit être satisfaite pour lier s à c_i et les degrés de liberté diminuent. Au fur et à mesure que nous avançons, nous ajoutons des contraintes, ce qui signifie géométriquement que nous réduisons progressivement les zones de l'espace où peuvent se trouver des éléments s valables (ceci peut être visualisé facilement dans un espace à deux dimensions) et nous améliorons par conséquent la liste des candidats (de l'ensemble S) à des appariements avec les c_i. Mais continuer avec cette logique, en affinant de manière itérative la liste des candidats à l'appariement et en faisant progressivement des choix, a un coût que nous avons pu mesurer avec l'algorithme précédent. Celui-ci débouchait sur une approche récursive fort longue, même si elle est déjà beaucoup plus efficace qu'une recherche exhaustive.

Pour réduire la longueur de la liste, nous proposons d'utiliser deux seuils. Premièrement, comme dans APPARIEV1, nous imposons un seuil – tol - qui fixe la différence relative tolérable au maximum entre les dissimilarités des éléments de la liste à s_1 et s_2 et les dissimilarités correspondantes de c_3 à c_1 et c_2. Cette contrainte permet d'éviter des appariements pauvres en éliminant des candidats à l'appariement qui donneraient dans S des configurations (s_1, s_2, s_3) différentes de celle observée dans C sur les trois premiers éléments (c_1, c_2, c_3). Mais la liste des candidats peut néanmoins rester longue. Pour la limiter, on impose une nouvelle contrainte - a priori - sur la longueur de cette liste : nous exigeons qu'elle soit inférieure à un nombre noté - env - (comme taille de l'environnement). Pour réduire la liste, il faudra donc éliminer des candidats potentiels de l'ensemble s à l'appariement avec c_3. Autant ne garder que les meilleurs. Pour chiffrer le caractère plus ou moins prometteur d'un candidat, on calculera la somme des différences absolues entre les paires correspondantes :

$$| \lambda \, d'(s, s_1) - d(c_3 , c_1)| + | \lambda \, d'(s, s_2) - d(c_3 , c_2)|$$

Plus cette somme est élevée, moins bon est l'appariement (à ce stade de l'algorithme) et moins bon est le candidat s. Il est cependant important de noter que cette sélection est dangereuse, car elle élimine des candidats de S sur base uniquement de la considération des éléments c_1 et c_2 de C. Une autre énumération (c_1, c_2, c_3 … c_n) des éléments de C pourrait déboucher sur d'autres résultats.

Malgré ces deux limitations le nombre total d'éléments retenus peut rester large (mais en tout cas le nombre est inférieur à env pour chaque liste de candidats à l'appariement avec c_i) car il ne prend en considération que la configuration dans C déterminée par les dissimilarités à c_1 et c_2.

L'idée supplémentaire exploitée dans la version deux de l'algorithme est simple. Cherchons à obtenir un premier ensemble d'appariements (disons des n premiers éléments de C) aussi bon que possible. Pour ce faire, nous utilisons l'algorithme simplifié précédent. Plus n sera grand, plus d'éléments s_i seront appariés à des c_i, plus le nombre de contraintes pour les appariements qui restent à faire sera grand, plus prometteurs seront les candidats en vertu du raisonnement proposé ci-dessus. Une fois ce premier ensemble d'appariements réalisé, basons-nous de nouveau sur la condition imposée par le troisième constat pour déterminer la liste des autres éléments de S à apparier. On constitue donc la liste des s appariables à c_{n+1}, sur base des contraintes de dissimilarités de s à s_1 ,s_2 , …, s_n , puis la liste des éléments appariables à c_{n+2} de la même manière et ainsi de suite. Nous conviendrons ensuite - et c'est ici qu'il y a une simplification drastique – d'arrêter l'itération et de ne garder dans chaque liste ainsi constituée qu'un seul élément pour chaque élément c_i de C (avec i > n). Nous choisirons le meilleur des éléments appariables en utilisant la mesure de qualité suggérée précédemment (comme une somme de différences en valeurs absolues entre les dissimilarités dans S et les dissimilarités dans C des éléments correspondants). Le paramètre n sera appelé la profondeur de la recherche. Si n est égal à N - l'effectif de l'ensemble cible - l'algorithme est équivalent à l'algorithme précédent (en prenant env suffisamment large). Si n est plus petit, l'algorithme ne va donc pas jusqu'au bout des itérations et se contente pour apparier les N-n éléments restants des meilleurs des éléments appariables au sens défini plus haut. S'il y a une solution exacte, l'algorithme permettra de la trouver.

8. Test de l'algorithme « APPARIEV2 »

De manière à pouvoir comparer les performances de cette nouvelle version de l'algorithme avec celles de la précédente, nous avons réutilisé les configurations cibles et les fichiers sources de la première simulation. Pour rappel, trois types de configurations cibles de 5 points ont été retenues avec un fichier source de 40 points dans le plan.

La version 2 utilise trois paramètres : un seuil– tol - comme dans la version 1- une taille maximum des listes des éléments appariables - env – et une profondeur de recherche notée ncomplet.

Le tableau ci-dessous donne pour quelques valeurs choisies de ces 3 paramètres les temps nécessaires en secondes pour trouver une solution.

Seuil (tol)	Environnement (env)	Profondeur (ncomplet)	Temps pour la configuration en ligne	Temps pour la configuration en carré	Temps pour la configuration en parabole
0.2	1	2	1.2	1.8	1.6
0.2	5	2	1.2	1.8	1.6
0.2	10	2	1.2	1.8	1.6
0.2	1	3	1. 3	2.0	1.9
0.2	5	3	1.3	2.3	2.3
0.2	10	3	1.3	2.3	2.3
0.2	1	4	1.3	2.1	2.0
0.2	5	4	1.4	2.4	2.7
0.2	10	4	1.4	2.4	2.7
0.8	1	2	3.2	3.8	3.8
0.8	5	2	3.2	3.8	3.8
0.8	10	2	3.2	3.8	3.8
0.8	1	3	4.7	6.7	6.6
0.8	5	3	10.9	18.1	17.8
0.8	10	3	18.8	30.4	29.4
0.8	1	4	5.4	8.4	8.3
0.8	5	4	29.8	59.4	56.6
0.8	10	4	92.0	175.2	161.0

TAB 3 – *Temps en secondes mis par APPARIEV2 pour trouver la solution en fonction des différents paramètres et du type de cible*

Il appelle plusieurs remarques.

Toutes les solutions optimales sont trouvées avec les valeurs de paramètres considérées, sauf pour le carré où il faut une profondeur de recherche au moins égale à 3 pour trouver la meilleure solution. Les solutions trouvées avec la profondeur 2 sont très proches de la solution optimale mais pas identiques.

Comme on pouvait s'y attendre, suivant la nature de la configuration cible, les temps d'exécution varient quasi du simple au double dans notre simulation.

Le paramètre env a surtout de l'importance lorsque le seuil est élevé. Ceci est compréhensible puisque lorsque le seuil est petit (0.2), les candidats à l'appariement, par définition,

sont limités et il n'y a pas lieu d'introduire une contrainte supplémentaire sur les listes de candidats.

Le seuil de tolérance reste, comme dans la simulation précédente, un paramètre clef, susceptible de réduire fortement les temps d'exécution.

Enfin, APPARIEV2 est beaucoup plus rapide. Testé sur l'appariement de photos sur des morceaux de musique décomposé en ensembles de mesures, il a pris entre 10 et 50 fois moins de temps qu'APPARIEV1 pour trouver la bonne solution. Les gains sont bien entendu fonction du choix des différents paramètres et trouver une bonne combinaison peut s'avérer laborieux.

Une dernière amélioration de l'algorithme est envisageable. Comme expliqué plus haut, tous les couples d'éléments possibles de l'ensemble source S sont tour à tour examinés et appariés aux deux premiers éléments de la cible (c_1, c_2). Dès qu'un couple a été choisi dans S, les autres éléments sont alors appariés progressivement : s_3 à c_3, s_4 à c_4, s_5 à c_5 et ainsi de suite ... L'ordre des éléments de la cible joue donc un rôle important. La figure 5 illustre ce constat dans un espace à deux dimensions. Comme les seuils sont exprimés en valeurs relatives, les zones sont définies par des anneaux superposés. Suivant qu'on commence avec le couple (c_1, c_2) (graphique de gauche) ou le couple (c_1, c_3) (graphique de droite) les zones (hachurées) de l'espace S (ici superposé à l'espace des éléments de C) où on peut trouver des éléments appariables diffèrent. Il est plus efficace de commencer avec des éléments de la cible aussi dispersés que possible, en les réordonnant si nécessaire.

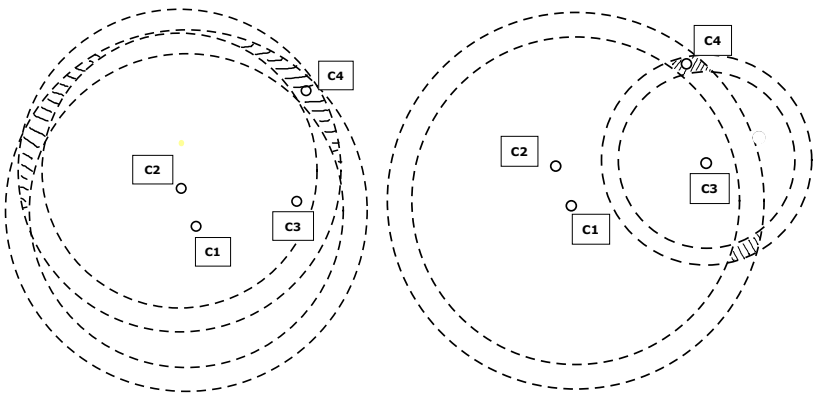

FIG. 5 - *Zones des éléments appariables à c_4 suivant qu'on considère d'abord (c_1, c_2) ou (c_1, c_3)*

9. Conclusions

Pour des ensembles de petites tailles, l'utilisation de APPARIEV1 et APPARIEV2 per-met de trouver des solutions dont on contrôle finement la qualité, ce qui est nécessaire dans certains problèmes. Mais dès que les tailles augmentent, APPARIEV0 se montre beaucoup plus performant. Son utilisation devrait permettre de s'attaquer à des banques de photos plus importantes pour construire des bandelettes musicales. APPARIEV2, moins rapide qu'APPARIEV0 et qui ne garantit pas de trouver la meilleure solution, n'est utile qu'à des fins d'exploration de l'espace des solutions « admissibles ».

Nos méthodes sont essentiellement combinatoires. Il est permis d'imaginer des angles d'attaque plus géométriques, en positionnant les éléments des ensembles S et C dans des espaces euclidiens au moyen d'analyses multidimensionnelles des proximités, de superposer les espaces puis avec des rotations et des dilations de l'espace Source de faire correspondre les ensembles d'éléments. C'est plus ou moins ce que l'on fait de manière intuitive lorsqu'on ajuste les formes « à l'œil ». Cette variante géométrique n'a pas été testée.

Références

Bosseur J.-Y. (2006), Musique et arts plastiques, *Interactions au XX^e siècle* , Minerve.

Defays D. (1978), A short note on a method of seriation, *British Journal of Mathematical Psychology* , vol. 31, pp. 49-53.

Defays D. (2017), Exploration on Trans-Sensory Mappings, à paraître dans *Art & Perception* .

Diday E. et Noirhomme-Fraiture M. (eds) (2008), *Symbolic Data Analysis and the SODAS Software* , Wiley.

Ox J. et Britton D. (2000), The 21st Century Virtual Reality Color Organ, *Journal IEEE Multimedia* , vol. 7, n° 3, pp. 6-9.

Pham N-K., Morin A., Gros P., Le Q.-T. (2009), Utilisation de l'analyse factorielle des correspondances pour la recherche d'images à grande échelle », Actes d'EGC, RNTI-E-15, *Revue des Nouvelles Technologies de l'Information - Série Extraction et Gestion des Connaissances*, Cépaduès Editions, pp. 283 - 294.

Widmer, G., Dixon S., Goebl W., Pampalk E., Tobudic A.(2003), In search of the Horowitz factor, *AI Magazine,* vol. 24, n° 3, pp. 111-130.