Facteurs en informatique musicale

Il est rappelé que le jury attend un exposé de 35 minutes, pédagogique et structuré, fondé sur ce sujet. Ce texte se conclut par une liste de pistes de réflexion pour guider la conception de l'exposé, mais toute initiative personnelle pertinente est appréciée. L'exposé doit contenir une ou plusieurs illustrations informatiques ainsi qu'une discussion autour d'une dimension éthique, sociétale, environnementale, économique ou juridique. Une piste particulière est proposée à cet effet.

1 Contexte

En 1798-1800, Beethoven écrit un quatuor à cordes (Op. 18, No 5) en imitant très clairement un quatuor de Mozart (K 464). Ce même quatuor, souvent cité comme un modèle de composition classique, est lui-même inspiré par un autre compositeur contemporain de Mozart, Haydn. En musique, les pastiches, ou compositions "à la manière de", consistent pour une artiste à écrire un morceau imitant le style d'une autre compositrice. L'imitation stylistique est aussi largement pratiquée dans d'autres disciplines artistiques : on peut par exemple citer Proust imitant dans *Pastiches et mélanges* plusieurs auteurs classiques dont Balzac et Flaubert, ou Franck Zappa pastichant la célèbre couverture de *Sgt. Pepper's Lonely Hearts Club Band*, des Beatles, qui montre un groupe de personnes devant un parterre de fleurs, avec une composition photographique similaire où les fleurs sont remplacées par des légumes.

L'imitation stylistique est réussie quand l'auditrice (la lectrice, la spectatrice...) parvient sans réfléchir à identifier le style copié. La question naturelle sous-jacente est la suivante : qu'est-ce qui caractérise un style artistique ? Dans sa version informatique, cette question devient : existe-t-il un modèle calculable (structure de données, ensemble de règles...), encodant le style d'une artiste donnée, et qui permette de produire artificiellement des oeuvres (images, textes, musique) que l'on puisse immédiatement attribuer à l'artiste ?

On étudie ici le coeur algorithmique d'un système d'improvisation musicale à style donné développé à l'IRCAM[1], qui permet à une musicienne, jouant un morceau *live*, d'improviser en duo avec un avatar d'elle-même. Le système enregistre la séquence sonore qui est jouée, en extrait les principaux éléments musicaux (segmentation en notes, durées, éléments de timbres), et construit avec ces derniers un générateur algorithmique imitant la musique jouée. Le générateur ne traite donc que de structures musicales symboliques, vues comme des mots sur un alphabet. Il peut produire, en temps réel, des séquences sonores, sur lesquelles les éléments rythmiques et de timbre sont resynthétisés.

[1]Institut de Recherche et Coordination Acoustique/Musique

Le système repose sur des algorithmes construisant, en temps réel, un modèle du style musical de l'artiste. Ce modèle doit également permettre de produire des séquences musicales imitant ce qui a été joué. Ce sujet s'intéresse à deux algorithmes qui ont été successivement intégrés au système de l'IRCAM pour générer les imitations musicales : l'algorithme de compression de Lempel-Ziv, présenté ici dans une version un peu inhabituelle sur des arbres, et l'algorithme de construction d'un oracle des facteurs. Ces deux algorithmes ont pour point commun d'encoder de façon compacte les/des facteurs d'un mot.

Dans la suite, on ne s'intéresse pas aux éléments sonores, ni rythmiques, du système : la séquence jouée est considérée comme un mot sur un alphabet Σ (pour fixer les idées, on peut imaginer que Σ est constitué des notations classiques des notes de musique do, do♯, ré, etc). Le mot vide est noté ε. La concaténation de deux mots w_1 et w_2 est notée $w_1 \cdot w_2$, ou $w_1 w_2$ lorsque cela n'entrave pas la compréhension. On note $|w|$ la longueur d'un mot w. Pour un mot w non vide et $i \in \mathbb{N}$, on note $\mathit{pref}_w(i)$ le préfixe de w de longueur i et $\mathit{suff}_w(i)$ le suffixe de w de longueur $|w| - i$ (on a donc $w = \mathit{pref}_w(i) \cdot \mathit{suff}_w(i)$).

2 Algorithme de Lempel-Ziv

Cette section décrit un algorithme de construction d'un arbre des préfixes d'un mot, exploitant le même principe que l'algorithme de compression de Lempel-Ziv.

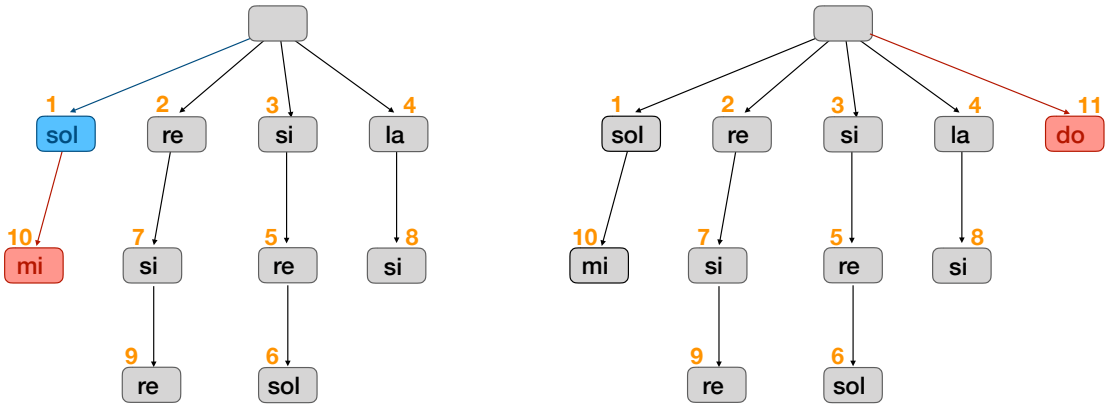
2.1 Construction de l'arbre

L'algorithme fonctionne en lisant le mot à encoder w, symbole par symbole. À un instant donné, on note u le préfixe de w déjà traité, et v le suffixe qui reste à traiter (avec $w = u \cdot v$). On construit l'arbre des préfixes en recherchant, dans l'arbre déjà partiellement construit sur u, le chemin correspondant au plus long préfixe possible de v, et en ajoutant un enfant à la suite de ce chemin.

On prend en exemple un alphabet simplifié de notes $\Sigma = \{do, re, mi, sol, la, si\}$ et le mot $w_0 = sol \cdot re \cdot si \cdot la \cdot si \cdot re \cdot si \cdot re \cdot sol \cdot re \cdot si \cdot la \cdot si \cdot re \cdot si \cdot re \cdot sol \cdot mi \cdot do$. Soit u_0 son préfixe de longueur 16.

L'arbre partiellement construit sur u_0 est donné par les noeuds gris et bleu de la Figure 1a. Pour continuer la construction, on lit le mot restant à encoder, soit v_0, symbole par symbole. Il existe déjà un chemin dans l'arbre sur son premiers symbole sol, on descend à la feuille correspondante (en bleu sur la figure 1a). En revanche, il n'existe pas de chemin incluant le symbole suivant mi : on arrête donc la lecture là, et on ajoute une nouvelle feuille étiquetée par mi (en rouge sur la figure 1a). Le nouveau mot restant est alors v_1 tel que $v_0 = sol \cdot mi \cdot v_1$. Son premier symbole est do, qui n'existe pas dans l'arbre. On l'ajoute donc directement à la racine (en rouge sur la figure 1b). C'est l'occasion de remarquer que la racine elle-même n'a pas d'étiquette : en effet, une telle étiquette serait un symbole présent sur tous les préfixes encodés par l'algorithme.

Pour construire l'arbre en entier, on commence avec un noeud racine vide, et on appelle récursivement la fonction décrite ci-dessus sur le mot w, privé de sa partie déjà encodée dans l'arbre. Pour pouvoir décompresser la séquence, il faut en plus marquer l'ordre dans lequel les préfixes ont été ajoutés dans l'arbre. Ces indices sont en orange sur la Figure 1b, qui donne l'arbre pour le préfixe w_0 de longueur 19.



(a) Ajout d'une feuille (en rouge) correspondant à la lecture de $sol \cdot mi$.

(b) Ajout d'une feuille (en rouge) correspondant à la lecture de do.

Figure 1: Ajout des premiers symboles du mot v_0, à l'arbre déjà construit sur u_0.

Le dernier chemin à traiter peut poser problème : si le dernier préfixe correspond déjà à un chemin dans l'arbre, qui ne nécessite pas l'ajout d'un nouveau nœud, il faudrait prévoir un cas particulier pour l'ajouter dans l'arbre. Cependant, dans un cas d'utilisation réel, les arbres sont suffisamment grands pour que l'on puisse s'autoriser à ignorer ce dernier chemin.

2.2 Re-génération à partir de l'arbre

Pour reconstruire la séquence initiale à partir de l'arbre des préfixes, il suffit de retrouver les chemins menant à chaque noeud, dans l'ordre dans lequel les noeuds ont été ajoutés. Si n est le nombre de noeuds de l'arbre, on applique pour i de 1 à n la méthode suivante : chercher le chemin menant au noeud numéroté i, ajouter à la séquence à reconstruire le mot formé par les symboles des noeuds de ce chemin.

Si maintenant on veut générer un mot qui ressemble au mot d'origine, mais ne lui est pas égal, on peut exploiter la structure de l'arbre : à l'étape i de reconstruction, avec une petite probabilité, on décide de modifier la séquence. Dans ce cas, supposons que l'on s'apprêtait à ajouter au mot en cours de construction un mot $v = v' \cdot \sigma$, où σ est l'étiquette du dernier noeud du chemin étiqueté par v. Soit k l'indice de ce même noeud. Le chemin étiqueté par v' amène au noeud parent de σ : si ce noeud a plusieurs enfants, alors on choisit un de ces enfants qui n'est pas σ, soit $\sigma' \neq \sigma$, et on ajoute $v' \cdot \sigma'$ à la séquence générée, au lieu de v. Dans ce cas, il faut aussi modifier les indices : au lieu de continuer avec l'indice $k+1$, on continue avec $k'+1$ où k' est l'indice du noeud contenant σ'.

Cette méthode génère une nouvelle séquence w'. Cependant, à chaque fois que la séquence est modifiée, le facteur v' a un préfixe commun au facteur v auquel il a été substitué, ce qui apporte une propriété importante musicalement : les modifications se produisent entre deux moments musicaux où le motif joué était identique. Cette propriété garantit, musicalement, une forme de cohérence.

On peut paramétrer cette méthode de différentes façons, par exemple en interdisant les modifications à moins d'une profondeur donnée, de façon à contraindre la longueur minimum des facteurs communs.

3 Oracle des facteurs

L'oracle des facteurs [1] d'un mot p est un automate fini, noté Oracle(p), acceptant tous les facteurs de p. Il est compact en mémoire, avec un nombre d'états **et** un nombre de transitions linéaires en la longueur de p. En outre, il en existe un algorithme de construction incrémental (appelé *online*). En contrepartie, l'oracle accepte trop de mots : il reconnaît tous les facteurs du mot d'origine, mais aussi des mots qui ne sont pas des facteurs.

Exemple 1 *On prend comme exemple dans la suite le mot* $p_0 = sol \cdot re \cdot si \cdot la \cdot si \cdot re \cdot si \cdot re \cdot sol$.

3.1 Automate de base

La construction de l'oracle se fait à partir d'un automate simple, qui reproduit le mot d'origine p. On le construit avec un état initial étiqueté 0, une suite d'états i pour i de 1 à $|p|$, et pour chaque état i, une transition vers l'état $i+1$ étiquetée par le i-ième symbole de p.

En outre, tous les états de cet automate sont finaux. Ainsi, l'automate de base accepte tous les préfixes de p. Cependant, il n'accepte pas les facteurs qui ne sont pas des préfixes.

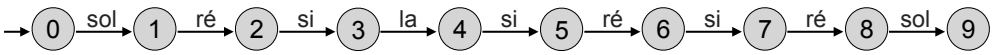


Figure 2: Automate de base du mot p_0. Tous les états sont finaux.

Exemple 2 *L'automate de base pour le mot* p_0 *est donné Figure 2. Il accepte bien les préfixes de* p_0*, mais le facteur* $re \cdot si \cdot re \cdot sol$*, par exemple, n'est pas accepté.*

L'oracle des facteurs est construit à partir de cet automate, en ajoutant astucieusement des transitions pour accepter aussi les facteurs non-préfixes.

3.2 Fonction S_p

La construction incrémentale de l'oracle des facteurs repose sur une fonction, appelée S_p, qui repère les répétitions dans p le mot en entrée. Cette fonction est définie des états de l'oracle dans les états de l'oracle. Pour i un état de Oracle(p), on s'intéresse à $pref_p(i)$, et on note v le plus long suffixe de $pref_p(i)$ qui apparaît au moins deux fois comme facteur de $pref_p(i)$. On pose alors $S_p(i) = j$ pour j l'état de Oracle(p) terminant la lecture de la première occurrence de v. Par convention, $S_p(0)$ vaut -1, et si v est le mot vide, $S_p(i)$ vaut 0.

Ainsi, la fonction S_p relie des états que l'on atteint avec un facteur commun le plus long possible.

Exemple 3 *Dans le mot* p_0*, les 4 premiers symboles apparaissent pour la première fois dans* p*, on a donc* $S_{p_0}(i) = 0$ *pour* $i = 1...4$*. Le 5e symbole, si, a déjà été vu en position 3, on a donc* $S_{p_0}(5) = 3$*. Pour l'état 8, le plus long suffixe répété de* $pref_p(8)$ *est* $si \cdot re$*, par les chemins* $4 \to 5 \to 6$ *et* $6 \to 7 \to 8$*, et* $S_p(8) = 6$.

3.3 Algorithme *online*

L'algorithme 1 est un fac-similé, traduit en français, de la version incrémentale de la construction de l'oracle, issue de l'article original [1]. On commence avec un automate constitué d'un seul état, initial, et étiqueté par 0 (fonction *Oracle_on_line* de l'algorithme 1). À chaque lecture d'un nouveau symbole σ, la fonction *add_letter* est appelée. Les lignes 2 et 3 servent à construire l'oracle de base décrit en 3.1.

```
1  Fonction add_letter(Oracle(p = p_1 p_2 ... p_m), σ)
2      Créer un nouvel état m + 1
3      Créer une transition de m à m + 1, labellisée par σ
4      k ← S_p(m)
5      tant que k > −1 et il n'existe pas de transition labellisée par σ depuis k
        faire
6          Créer une transition de k à m + 1 labellisée par σ
7          k ← S_p(k)
8      si k = −1 alors
9          s ← 0
10     sinon
11         s ← l'état où mène la transition depuis k par σ
12     S_pσ(m + 1) ← s
13     retourner Oracle(p = p_1 p_2 ... p_m, σ)
14 Fonction Oracle_on_line(p = p_1 p_2 ... p_m)
15     Créer Oracle(ε) avec un seul état 0
16     S_ε(0) ← −1
17     pour i ← 1 à m faire
18         Oracle(p = p_1 p_2 ... p_i) ← add_letter(Oracle(p = p_1 p_2 ... p_{i−1}),p_i)
```

Algorithme 1 : Construction d'un oracle des facteurs, reproduit de [1], sur p un mot. Le i-ième symbole de p est noté p_i.

Il faut ensuite ajouter des transitions ayant un rôle de raccourci dans le mot, de façon à accepter des facteurs non-préfixes. Pour cela, on s'appuie sur la fonction S_p. À l'étape m, la construction repose sur un parcours du *chemin des suffixes* de l'état courant, défini par l'ensemble des états k_i où $k_{i+1} = S_p(k_i)$, avec $k_0 = m$. Pour tous les états du chemin des suffixes (boucle ligne 5 de l'algorithme 1), on ajoute une transition étiquetée par le symbole courant σ vers l'état $m + 1$ (ligne 6), s'il n'existait pas déjà une transition en σ.

Ces transitions de raccourci permettent d'accepter des facteurs qui ne sont pas préfixes, ainsi que, dans certains cas, des mots qui ne sont pas des facteurs. En effet, si l'on admet que les facteurs de $pref_p(m)$ sont reconnus par l'automate partiellement construit à l'étape m, il suffit d'ajouter des transitions reconnaissant les facteurs de la forme $v \cdot \sigma$, pour v un facteur suffixe de $pref_p(m)$. Ces facteurs sont, justement, ceux du chemin des suffixes. En ajoutant une transition σ depuis les états de ce chemin, jusqu'à $m + 1$, on accepte bien les facteurs de la forme $v \cdot \sigma$. Dans la suite, on admet que l'oracle des facteurs d'un mot p accepte au moins les facteurs de p.

Exemple 4 *La figure 3 donne l'oracle des facteurs du mot* p_0. *On reconnaît l'automate*

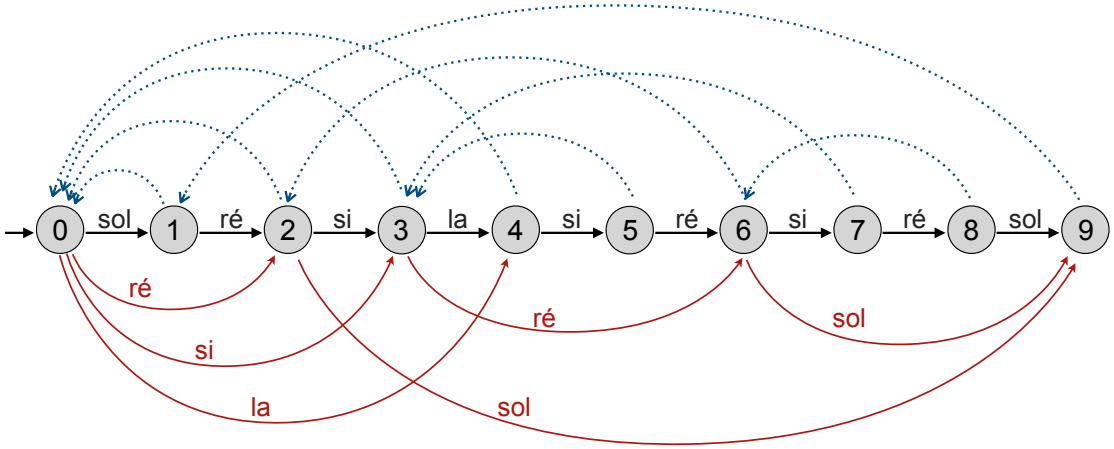


Figure 3: Oracle des facteurs du mot p_0 donné en exemple. Les flèches en pointillés bleus montrent la fonction S_{p_0}.

de base de la figure 2 dont les transitions sont en noir. Les transitions de raccourci sont en rouge, et les flèches en pointillés bleus représentent la fonction S_{p_0}.

À l'appel de *add_letter*, il faut aussi effectuer le calcul de S_p pour l'état $m+1$, qui vient d'être ajouté (lignes 8 à 12 de l'algorithme 1). La boucle ligne 5 suit le chemin des suffixes jusqu'à un état k qui, soit avait déjà une transition sortante en σ, auquel cas on donne à $S_p(m+1)$ la valeur de l'état où mène cette transition depuis k (ligne 9), soit est l'état initial, auquel cas $S_p(m+1)$ vaut 0 (ligne 11).

Exemple 5 *Considérons l'état* 9 *de l'automate de la figure 3. Lorsqu'il a été ajouté, on avait* $m = 8$ *et* $\sigma = sol$. *Le chemin des suffixes depuis l'état* 8 *emmène d'abord dans l'état* 6, *qui partage avec l'état* 8 *le chemin entrant* $si \cdot re$, *puis* 2, *qui partage le chemin* re, *puis* 0. *Les états* 6 *et* 8 *n'ayant pas de transitions sortante en sol, les transitions de raccourci* $6 \xrightarrow{sol} 9$ *et* $2 \xrightarrow{sol} 9$ *sont ajoutées. Elles permettent de reconnaître les facteurs terminant par sol, par exemple* $re \cdot si \cdot re \cdot sol$ *(en passant par les états* 2 *et* 6*), ou bien le facteur* $re \cdot sol$ *(en passant par l'état* 2*). L'état* 0, *lui, a une transition sortante en sol, ce qui arrête le parcours du chemin. On pose donc* $S_{p_0}(9) = 1$.

3.4 Re-génération à partir de l'oracle

La méthode décrite ici est extraite de [2]. On remarque que, par construction, les transitions ajoutées à la ligne 3 (en noir sur la figure 3) de la fonction *add_letter* sont celles de l'automate de base, qui reproduisent le mot p à l'identique. De même que pour l'arbre des préfixes, la re-génération de nouvelles séquences consiste à suivre par défaut ces transitions, et à modifier cet ordre avec une petite probabilité.

La modification exploite la fonction S_p, qui a une propriété intéressante musicalement : elle relie des états ayant un facteur entrant identique. Musicalement, ce sont des instants où un même motif (suite de notes) a été joué, donc, probablement, des instants partageant une même couleur musicale. Les modifications se produisent donc entre des états reliés par S_p. Pour modifier la séquence, si on est dans un état k, on considère $j = S_p(k)$. Il faut qu'elle soit strictement positive (sinon, le facteur entrant commun est ε, ce qui n'est pas intéressant musicalement). On se place alors

dans l'état j, on pose P_j l'ensemble des transitions sortantes de l'état j et on tire aléatoirement une transition dans P_j, qui est choisie pour continuer la génération.

Exemple 6 *Supposons être arrivée dans l'état* 7 *au moment où on décide de modifier la reconstruction. On vient donc de jouer, par les transitions normales,* $re \cdot si$. *On suit* $S_{p_0}(7)$ *pour arriver dans l'état* 3*, dans lequel on arrive aussi, normalement, avec* $re \cdot si$. *On a* $P_7 = \{3 \xrightarrow{la} 4, 3 \xrightarrow{re} 6\}$. *On pourra donc continuer soit avec un* la *dans l'état* 4*, soit avec un* re *dans l'état* 6. *Dans les deux cas, les motifs* $re \cdot si \cdot la$ *et* $re \cdot si \cdot re$ *avaient été jouées dans le mot d'origine* p_0.

L'oracle des facteurs est utilisé dans un système d'improvisation à style donné développé à l'IRCAM depuis 2006, notamment en situation de concert. En concert, la musicienne joue un air, capté par le système et séquencé en symboles/notes de musique. Pendant qu'elle joue, l'oracle se construit incrémentalement, note par note. Dès qu'il est suffisamment gros pour devenir intéressant, on peut commencer à produire une séquence musicale inspirée de l'air joué par la musicienne. Évidemment, cette séquence est aussi écoutée par la musicienne, qui va en tenir compte dans sa propre improvisation : elle se retrouve, en quelque sorte, dans une situation d'improvisation en duo avec elle-même.

4 Suggestions pour le développement

1. Présenter la construction de l'arbre de la Section 2 et de l'oracle de la Section 3 sur le mot donné en exemple, ou un autre mot de votre choix (pas nécessairement sur le même alphabet). Proposer des re-générations à partir de l'arbre/oracle construit.

2. Dans un contexte de génération musicale à style donné, que se passe-t-il avec l'algorithme de Lempel-Ziv si une modification intervient à la racine de l'arbre ? Une telle situation peut-elle se produire avec l'oracle des facteurs ? Discuter des limites des deux algorithmes et des améliorations qui peuvent leur être apportées.

3. Discuter les questions de complexité pour une utilisation dans un contexte temps réel, tel que l'improvisation musicale *live*, des algorithmes de la Section 2, et/ou de l'automate produit Section 3.

4. Imaginons que vous soyez développeur d'un système d'improvisation automatique reposant sur l'un ou l'autre des algorithmes décrits ici. Vous recevez la visite d'une productrice qui vous propose de participer à la production d'un concert évènement, avec le chanteur français Aldebert, qui jouerait en duo avec des avatars que vous devrez créer à partir de sons collectés en ligne : en première partie, de la chanteuse contemporaine Beyoncé Knowles, et en deuxième partie, du compositeur et pianiste Franz Liszt (1811-1886). Que lui répondez-vous ?

5. Implémenter les algorithmes d'encodage, décodage et regénération de la Section 2 dans un langage de votre choix, présenter l'implémentation.

6. Construire l'oracle des facteurs du mot *abbbaab*, et calculer les mots reconnus. Construire un automate déterministe n'acceptant que les facteurs de *abbbaab*. Caractériser les cas où l'oracle accepte des mots qui ne sont pas des facteurs du mot d'origine.

7. Comment caractériser le modèle du style musical, implémenté en pratique par les deux algorithmes présentés ? Sur quel type d'information repose-t-elle ? Quel type d'infomation lui échappe ?

References

[1] Cyril Allauzen, Maxime Crochemore, Mathieu Raffinot. *Factor Oracle: A New Structure for Pattern Matching*, SOFSEM, 1999, pp 295-310.

[2] Gérard Assayag, Shlomo Dubnov, *Using Factor Oracles for Machine Improvisation*, Soft Computing, Vol. 8, Nb 9, pp 604–610, 2004.