Documentation de Soracle

Merlin Chao

juin 2025

Table des matières

1 Introduction **1**
1.1 Contributions 2
1.2 Aperçu du modele 2

2 description du modèle **2**
2.1 L'oracle des facteurs 2
2.2 Navigation 3
2.3 Choix des candidats 4

3 Implémentation **5**
3.1 Environnement Python 5
3.2 Patchers Max 5

4 Guide d'utilisation **6**
4.1 Paramètres du Player 6
4.2 Transpositions 9
4.3 Chargement et sauvegarde d'oracles 9
4.4 Multisegmentation 9

5 Conclusion et perspectives **9**

1 Introduction

Soracle est une extension de Somax2. Il conserve toutes les fonctionnalités de Somax2 et introduit un nouveau modèle de navigation basé sur l'oracle de facteurs. Pour chaque corpus, un oracle est construit automatiquement. Le *Player* utilise les liens suffixes pour naviguer à l'intérieur du corpus selon des stratégies de saut que nous détaillerons dans la section 4. Ces stratégies s'inspirent principalement de l'approche décrite dans [1].

1.1 Contributions

Bien que l'oracle ait déjà été employé dans d'autres logiciels de la galaxie Omax (Omax, Dicy2 ...), Soracle se distingue par plusieurs aspects :

1. Soracle filtre ses candidats par influence, devenant ainsi un «Omax qui écoute».
2. Il utilise simultanément plusieurs oracles (actuellement un pour le pitch et un pour le chroma).
3. Intégration complète à Somax2 : il est possible de basculer entre l'oracle et le modèle original à la volée.
4. Compatibilité avec Omax : les oracles générés par Soracle peuvent être exportés et réutilisés dans Omax5 (fonctionnalité expérimentale).

1.2 Aperçu du modele

Pour chaque corpus, deux oracles sont construits : l'un à partir des valeurs de pitch, l'autre à partir des vecteurs chroma. Les événements sont d'abord joués de façon linéaire pendant un nombre prédéterminé d'événements, puis un saut est effectué en recherchant un événement possédant un contexte commun avec l'événement courant. Les deux oracles sont parcourus en parallèle, et les candidats trouvés sont pondérés en fonction de leur oracle d'appartenance(pitch ou chroma). Une fenêtre autour du point de saut (*window size*) étend la recherche pour augmenter le nombre de candidats.

De plus, lorsque le *Player* reçoit des influences externes, les candidats sont filtrés pour ne conserver que ceux dont le pitch ou le chroma correspondent à la dernière influence reçue. Les scores des candidats sont ensuite réajustés pour favoriser ceux qui matchent avec les features de l'influence.

2 description du modèle

2.1 L'oracle des facteurs

L'oracle est un automate fini construit à partir d'une séquence d'entrée [3], [4]Il encode efficacement tous les facteurs de cette séquence, et une recherche rapide de motifs similaires.La figure 1 montre un oracle construit avec le mot abcbacbaba. Dans notre cas, les oracles seront construits à partir des labels des features (pitch ou chroma) du corpus.

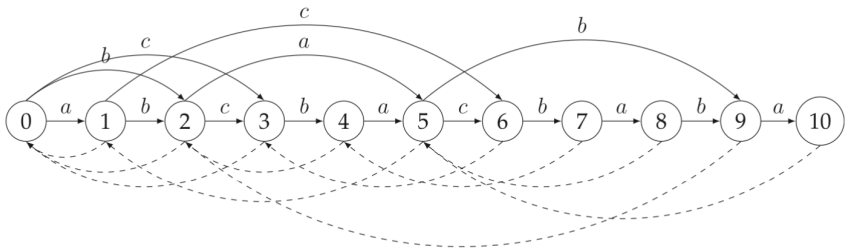

FIGURE 1 – Exemple d'un oracle construit pour le mot w = abcbacbaba. Les lignes pleines sont les transitions et les pointillés sont les liens suffixes.Ce schéma est tiré de [5]

L'oracle est composé d'un ensemble d'états (un par élément de la séquence), reliés par plusieurs types de transitions. Les transitions qui nous intéressent sont les liens suffixes (suffix links) : ils relient un état i à l'état j correspondant au plus long suffixe répété de la séquence allant jusqu'à i.

Pour identifier des points du corpus partageant un contexte similaire au moment courant, nous exploitons une structure dérivée du graphe :

- Les **suffix links**, qui permettent de remonter la structure répétitive du corpus.
- Les **reverse suffix links**, c'est-à-dire les suffix links considérés sans direction : chaque lien est exploité dans les deux sens.
- Les **reverse suffix links des cibles** des suffix links : pour chaque état, on explore aussi les reverse suffix links de l'état atteint via son suffix link, étendant ainsi la recherche à des contextes voisins dans la structure du corpus.

Cette combinaison permet d'extraire un ensemble d'états candidats ayant des contextes similaires.

2.2 Navigation

La stratégie de navigation est globalement tirée de [1]. Une phase de continuité de longueur fixée N est imposée, durant laquelle l'agent rejoue N éléments consécutifs du corpus. Ce n'est qu'après cette séquence que la recherche d'un saut discontigu est autorisée. Afin d'augmenter le nombre de candidats possibles pour ce saut, une fenêtre de recherche de taille W est appliquée autour du point de saut. Si i désigne l'indice de l'état atteint après le dernier saut, alors les prochains candidats au saut sont recherchés dans l'intervalle $[i+N-W, i+N+W]$ du corpus.

2.3 Choix des candidats

Soient $\mathcal{O}_p$ et $\mathcal{O}_c$ les oracles respectifs de *pitch* et de *chroma* associés à un corpus. À un instant t, chaque oracle $\mathcal{O}_f$ (avec $f \in \{pitch, chroma\}$) fournit un ensemble de candidats $C_f(t) = \{c_{f,1}, \ldots, c_{f,n_f}\}$, chacun étant associé à un score $s(c_{f,i}) \in [0,1]$, basé sur le *longest repeated suffix* (LRS) normalisé.Les candidats qui ont un LRS inférieur au seuil limite sont automatiquement enlevés.

On forme ensuite l'ensemble total des candidats par union des ensembles individuels :

$$C_{total}(t) = C_{pitch}(t) \cup C_{chroma}(t).$$

Un score *interne* est attribué à chaque candidat $c_i \in C_{total}(t)$ selon les poids $w_{int,f} \in [0,1]$ affectés à chaque oracle :

$$s_{int}(c_i) = \sum_{f \in \{pitch, chroma\}} w_{int,f} \cdot I_{c_i \in C_f} \cdot s(c_i).$$

Si le système reçoit une influence externe, on note $p(t)$ et $c(t)$ les valeurs de pitch et chroma perçues à l'instant t. On applique alors un filtrage basé sur cette influence :

$$s_{ext}(c_i) = w_{ext,p} \cdot I_{pitch(c_i)=p(t)} + w_{ext,c} \cdot I_{chroma(c_i)=c(t)},$$

où $w_{ext,p}, w_{ext,c} \in [0,1]$ sont les poids des filtres pour le pitch et le chroma.

On obtient alors le score final d'un candidat :

$$S(c_i) = s_{int}(c_i) \cdot s_{ext}(c_i).$$

Si le corpus n'a pas de notes en commun avec les influences qu'il reçoit,alors l'oracle ne trouvera aucun candidat. Pour remédier à ce problème, on autorise la transposition des événements. Le principe est expliqué plus en détail plus loin.

Une distribution de probabilité sur l'ensemble des candidats est ensuite obtenue via une softmax, régulée par une température $T > 0$:

$$P(c_i) = \frac{\exp\left(\frac{S(c_i)}{T}\right)}{\sum_j \exp\left(\frac{S(c_j)}{T}\right)}.$$

Le candidat est alors sélectionné par tirage probabiliste selon cette loi.

.

3 Implémentation

3.1 Environnement Python

J'utilise la librairie Python VMO [2] pour construire les oracles. Les scripts dédiés à Soracle se trouvent dans le dossier :

`python/somax/factor_oracles`

La classe principale est `vmo_Player`. C'est une classe utilisée par `somax2.Player`,pour gérer l'utilisation des oracles pour la navigation. Lorsque le `vmo_Player` est activé, le `somax2.Player` utilisera le modèle de soracle pour donner un nouvel événement en output de sa fonction `new_event`. Le `somax2.Player` continue de mettre à jour les `atoms`, donc on peut revenir au modèle original de Somax2 avec fluidité.

Les principales classes associées sont :

- `vmo_manager` : parcourt les oracles et trouve des candidats pour les sauts.
- `navigator` : paramètre la navigation (nombre d'événements avant saut, taille de la fenêtre, etc.).
- `influence_handler` : filtre les événements en fonction des influences externes.

3.2 Patchers Max

Les patchers pour Soracle sont situés dans :

`max/somax/Soracle`

L'objet principal est `soracle.Player`, décliné en deux interfaces : `soracle.Player.ui` et `soracle.Player.app`, à l'instar de `somax.Player`.

4 Guide d'utilisation

4.1 Paramètres du Player

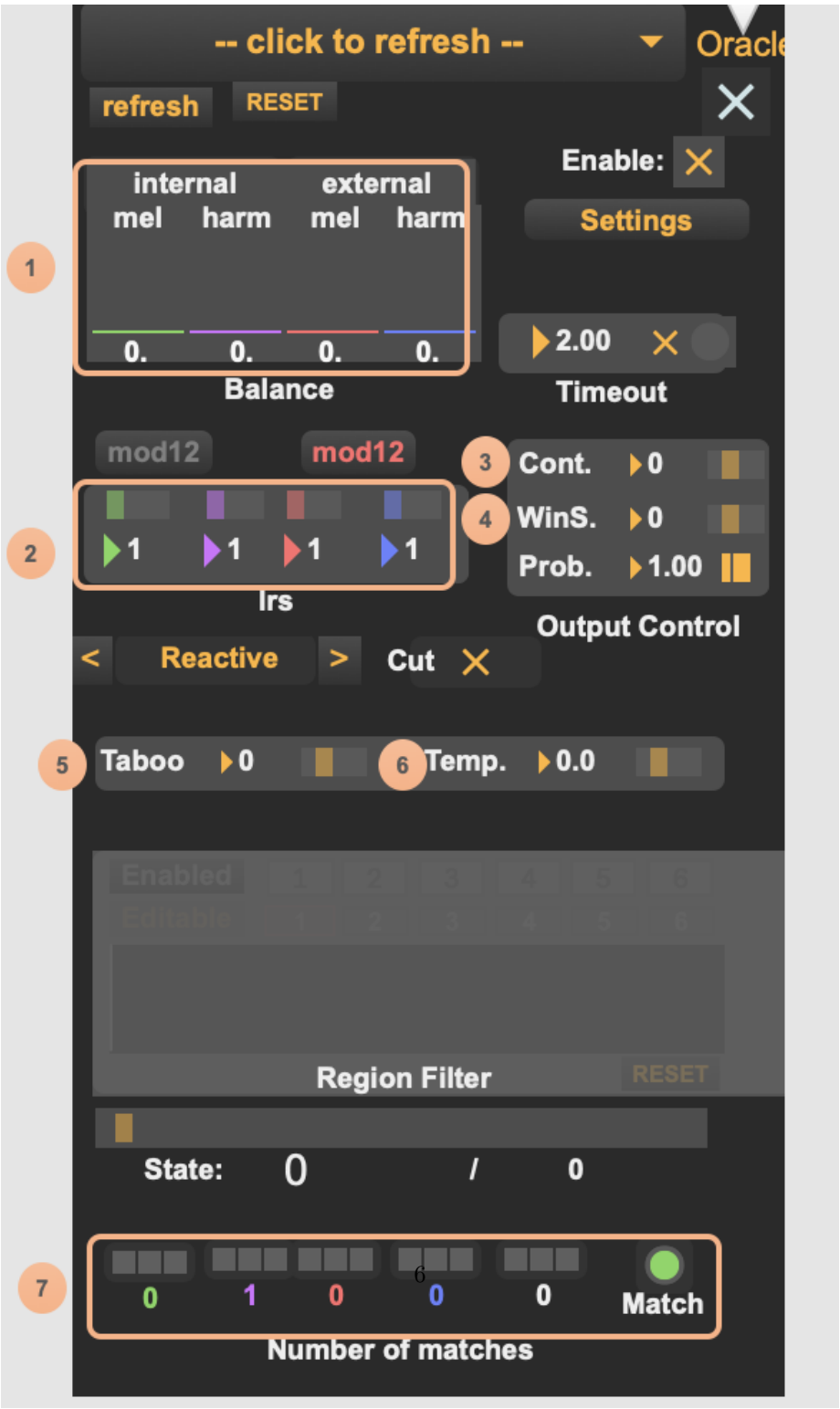

FIGURE 2 – Interface utilisateur du Player

Les contrôles hérités de Somax2 (Timeout, probabilités, réaction/continu, cut) conservent leur fonction originale et ont la même implémentation. Les nouveaux paramètres sont :

1. **Poids externes** : pondèrent les candidats qui matchent avec les influences (pitch ou chroma). Si un candidat matche plusieurs influences, les poids sont additionnés.
   Poids internes : équilibrent l'importance relative des oracles pitch et chroma. Les scores des candidats sont leurs LRS multipliés par le poids de l'oracle les ayant générés.
2. **LRS interne/externe** : seuils minimum de similarité pour filtrer les candidats (interne : au sein d'un oracle ; externe : par rapport aux influences).
3. **Continuité** : nombre d'événements consécutifs joués avant de déclencher une nouvelle recherche de saut.
4. **Window Size (WinS)** : nombre d'événements en amont et en aval du point de saut considérés comme candidats. Une valeur élevée améliore la diversité, mais rend la navigation moins prévisible.
5. **Taboo** : Le taboo fonctionne de la même manière que dans Somax2 mais c'est une autre implementation.
6. **Température** : paramètre de la distribution softmax pour le choix parmi les candidats. Une température basse favorise les scores élevés, une température élevée uniformise la sélection.
7. **Visualisation des matches** : Affiche les sauts à venir. Les external matches sont toujours en nombre inférieur aux internal matches, car ils sont filtrés par l'influence. Le nombre total de matches correspond (pour l'instant) à la somme des deux types combinés, ce qui signifie qu'un même match peut être compté plusieurs fois. Cette valeur n'est donc pas totalement fiable.

FIGURE 3 – Paramètres du Player

4.2 Transpositions

Pour chaque transposition, l'algorithme cherche dans l'oracle les contextes identiques mais transposés du corpus actuel. On commence par des contextes d'un événement et on étend jusqu'à ne plus trouver de correspondance. Cette méthode a l'intérêt de n'utiliser qu'un seul oracle pour toutes les tonalités.

4.3 Chargement et sauvegarde d'oracles

Le bouton `Load Oracle` charge un oracle existant, et `Save Oracle` exporte l'oracle courant. Ces fichiers peuvent être réutilisés dans Soracle et Omax5. Pour l'instant, l'intérêt est limité dans Soracle car on ne peut pas customiser les oracles, mais cela peut au moins avoir l'intérêt de réduire le temps de chargement.Pour désactiver la création automatique à chaque chargement de corpus, décochez `Auto Oracle Creation`.

4.4 Multisegmentation

Le principe de la multisegmentation est d'utiliser une segmentation distincte pour les oracles de chroma et de pitch, basée sur les variations de ces caractéristiques. On construit deux corpus segmentés selon la norme des différences de la feature, puis les deux oracles correspondants. Comme les événements ne coïncident plus, la navigation se fait en temps (secondes) plutôt qu'en index d'événements.

Si deux candidats proposés par les deux oracles sont très proches en temps (< 50 ms), ils sont considérés identiques et leurs scores sont additionnés. Lorsqu'un candidat est choisi, le Player continue de jouer selon la segmentation du corpus choisi pendant une durée égale au paramètre de continuité (en secondes).

Il y a un exemple dans le patcher `soracle_tuto.maxpat`.Voici le résumé : créez deux corpus avec `soracle.audiocorpusbuilder`, nommés `Chroma` et `Pitch`, placez-les dans le même dossier, puis sélectionnez ce dossier avec le bouton `Load Multi-Seg` du Player.

5 Conclusion et perspectives

L'utilisation de l'oracle permet une improvisation généralement plus structurée que celle offerte par Somax2, notamment lorsque le player fonctionne en mode continu. Soracle offre également à l'opérateur un meilleur contrôle sur la continuité de l'improvisation, en permettant de définir précisément le nombre d'événements consécutifs. En comparaison, le paramètre de continuité dans Somax2 peut se révéler assez imprévisible d'un corpus à l'autre.

Le principal point faible de Soracle, à ce jour, concerne l'écoute : celle-ci se limite à un simple filtre sur la note d'arrivée du saut, ce qui est nettement moins performant que l'approche de Somax2. Il pourrait donc être intéressant d'envisager un modèle hybride, combinant l'oracle pour la navigation interne au corpus, et Somax2 pour l'écoute des influences.

Références

[1] Gérard Assayag and Georges Bloch, *Navigating the Oracle : a heuristic approach*, in Proc. Intl. Computer Music Conf., 2007.

[2] Cheng-I Wang, Jennifer Hsu,Shlomo Dubnov *Machine Improvisation with Variable Markov Oracle : Toward Guided and Structured Improvisation*, in Computers in Entertainment (CIE), 2016

[3] Allauzen, C., Crochemore, M., Raffinot *Factor Oracle : A New Structure for Pattern Matching*, In G. Tel et al., eds. SOFSEM'99 : Theory and Practice of Informatics. Berlin : Springer, pp. 291–306.

[4] Lefebvre, A., and T. Lecroq. *Computing Repeated Factors with a Factor Oracle*, In Proceedings of the Australasian Workshop On Combinatorial Algorithms, pp. 145–158.

[5] K. Déguernel, E. Vincent and G. Assayag *Probabilistic Factor Oracles for Multidimensional Machine Improvisation*, in Computer Music Journal, vol. 42, no. 02, pp. 52-66, June 2018