Un modèle pour la gestion des séquences temporelles synchronisées. Application aux données musicales symboliques.

Zoé Faget

Université Paris-Dauphine

N° attribué par la bibliothèque

THÈSE

pour obtenir le grade de

Docteur de Université Paris-Dauphine

Spécialité : **Data Management**

préparée au laboratoire **LAMSADE**

dans le cadre de l'École Doctorale **EDDIMO**

présentée et soutenue publiquement
par

Zoé Faget

le 6 décembre 2011

Titre:

Un modèle pour la gestion des séquences temporelles synchronisées. Application aux données musicales symboliques

Directeur de thèse: **Philippe Rigaux, CNAM**

Jury

M. Dominique Laurent,	Univ. Cergy-Pontoise, Rapporteur
M. Bruno Defude,	Télécom Paris-Sud, Rapporteur
M. Cédric du Mouza,	CNAM
M. Geneviève Jomier,	Univ. Paris-Dauphine
M. Philippe Rigaux, CNAM,	Directeur de thèse

Table des matières

Bibliographie 91

Remerciements...

Tout d'abord je remercie les rapporteurs Bruno Defude et Dominique Laurent qui ont accepté de consacrer une partie de leur temps à l'étude de mon travail.

Je remercie également Geneviève Jomier et Cédric Du Mouza qui ont, à deux reprises, fait partie de mon jury.

Des remerciements chaleureux pour David Gross-Amblard, Virginie Thion-Goasdoué et Camélia Constantin pour leur gentillesse, leur disponibilité, et pour les conversations fructueuses que j'ai pu avoir avec eux qui ont contribué à l'avancée de mon travail.

Un merci spécial à mon ami Benoît, je sais combien je lui dois.

La société Armadillo m'a accueillie comme une famille pendant ces trois années.

Enfin et surtout, je remercie mon directeur Philippe Rigaux pour son encadrement, son écoute, ses conseils, sa patience et sa confiance.

Merci à tous.

Résumé de la thèse

Mots clés : Modèle de données, séquences temporelles, recherche d'information musicale, bibliothèques numériques, recherche par contenu, index n-gram, signatures algébriques.

La première partie de ma thèse est la description d'un modèle algébrique pour la gestion des séquences temporelles synchronisées. Ce modèle est une extension du modèle relationnel classique auquel on ajoute un type nouveau, le type séquence temporelle. L'algèbre relationnelle est augmentée de trois opérateurs dédiés à ce nouveau type. Ces opérateurs permettent de retrouver toutes les opérations classiquement conduites sur des séquences temporelles. Le langage utilisateur correspondant est exposé, ainsi que de nombreux exemples, puisés notamment dans le domaine de la gestion des partitions symboliques.

La seconde partie est la description d'un index permettant de réaliser plusieurs types de recherches dans des partitions symboliques (exacte, transposée, avec ou sans rythme et approchée). Il repose notamment sur la notion de signature algébrique.

Dans la dernière partie, je décris une plateforme dédiée à la gestion du contenu musical symbolique qui est une application des deux précédentes parties.

Keywords : Data model, time series, music information retrieval, digital libraries, content-based retrieval, n-gram indexing, algebraic signatures.

The first part of my thesis is the description of an algebraic model for synchronized time series. It's an extension of the classic relational model, to which we add a new type : the time series type. The relational algebra is enhanced with three new operators dedicated to time series. Those operators enable us to retrieve all usual operations on time series. A user query language is provided, as well as several examples, most of which are chosen in the symbolic score management domain.

The second part is the description of an index structure allowing to conduct several types of queries on symbolic scores (exact, transposed, with or without rythm and approximate). This index relies strongly on the notion of algebraic signature.

In the last part, I describe a platform dedicated to symbolic music scores management, which is an application of the two previous parts.

Introduction

La musique est de plus en plus présente sur le web, et son accès est rendu possible par une multiplication des plateformes. On observe entre autres la diversification des représentations, la multiplication des outils d'étude et d'interrogation, et une évolution imprévisible des besoins et des types d'utilisations.

Proposer une démarche générique pour la gestion de contenu musical permet de faire face à ces changements constants et aux problématiques qui leurs sont liées.

1 Contexte et énoncé des problématiques

Historiquement, les premiers utilisateurs de contenu musical sur internet étaient *amateurs* de musique, recherchant une consommation musicale immédiate (écoute ou téléchargement). Les fournisseurs de contenu étaient quant à eux des professionnels de type commercial (vente de musique) ou artistique (musiciens proposant leurs créations). La multitude de formats audio extrêmement compacts a toujours facilité la diffusion de la musique sous cette forme.

Depuis quelques temps, de nouvelles communautés d'utilisateurs *experts* apparaissent : musicologues, éditeurs de musique, musiciens, historiens de la musique. Ces utilisateurs se distinguent des amateurs par des besoins très différents. Parmi ces besoins, on distingue :

- le partage et la mise en correspondance d'archives visant à faire émerger des connaissances nouvelles et des analyses portant sur des corpus plus étendus ;
- l'ouverture au public de contenus autrement restreints à un cercle confidentiel et initié (se posent alors les problématiques de *copyright*, sécurité et contrôle de la distribution) ;
- l'étude de corpus intéressants pour leur valeur historique ou pour leur rareté plus que pour leur qualité esthétique ;
- la création des corpus de documents dont la diffusion n'est possible que numériquement ;
- la traduction numérique de l'ensemble des outils d'analyse musicologique traditionnels.

Ce dernier point en particulier présente un enjeu important pour les musicologues. En effet, au-delà des outils classiques de recherche (exacte, par motifs, par similarité, transverses) et de parcours, il est nécessaire d'intégrer aux nouvelles possibilités technologiques les techniques d'analyse existantes développées par les musicologues depuis des siècles.

À cette fin, la représentation audio, de loin la plus courante, est très mal adaptée. Il est en effet quasi-impossible d'en extraire une description objective. On choisit

donc une représentation dite *symbolique* qui met l'accent sur l'aspect structuré du contenu. Si la subjectivité de la musique (interprétation, ressenti) est ignorée dans ce type de représentation, en revanche un grand nombre d'informations traditionnellement difficiles à extraire d'un enregistrement audio deviennent accessibles.

Définition 1. *(Données musicales symboliques) On appelle* données musicales symboliques *la description détaillée de toutes les informations nécessaires à l'affichage (ou gravure) précis d'une partition.*

On y trouve ainsi les hauteurs et durées des notes, les différents instruments, les paroles, mais également d'autre types d'informations comme doigtés, nuances, tempo, voire des propriétés de mise en page.

Une partition sous sa forme symbolique contient donc un nombre important d'informations hétérogènes. Dans un contexte applicatif, on regroupe les partitions en *collections*.

On appelle *collection* un ensemble de partitions ayant un certain nombre de caractéristiques communes. Ces caractéristiques peuvent être de type formelles, historiques, géographiques, stylistiques ou autre. L'intérêt et les études conduites varient grandement d'une collection à l'autre.

On cherche donc à concevoir des outils pouvant s'adresser à de grandes collections de partitions symboliques et se décliner en fonction de leurs caractéristiques propres. C'est le premier objectif de cette thèse.

Lorsque ces collections grandissent, une nouvelle problématique apparaît. Les informations présentes sont très riches et les données à interroger deviennent rapidement très volumineuses. Il est nécessaire d'indexer de manière intelligente et performante ce contenu hétérogène, complexe et non-conventionnel. Construire au cas par cas les opérations spécifiques au contenu indexé à partir d'une méthode d'indexation classique est contre-productif et instable. D'une part, chaque enrichissement de la collection par de nouvelles partitions de structures différentes peut rendre l'index inutilisable si celui ci n'est pas conçu de manière générique. D'autre part, un contenu musical symbolique peut être interrogé de multiples manières : avec ou sans rythme, transposé ou non, par similarité, par motif. Ces différents types d'interrogations portent sur des informations distinctes contenues dans la partition, mais qui peuvent se recouper d'un type d'interrogation à l'autre. Un seul index permettant de satisfaire toutes ces requêtes représente un gain de temps et d'espace.

La conception d'un index compact, flexible et performant est le second objectif de cette thèse.

En résumé, représenter autant qu'indexer les données musicales symboliques représente un défi important. Le contenu structuré d'une partition symbolique propose une information très riche mais spécifique à chaque collection. Notre but est de proposer le cadre de travail le plus ouvert possible pour répondre aux besoins de manière globale et pérenne.

2 Contenu et organisation du mémoire

En réponse à notre premier objectif, nous définissons un modèle logique destiné à la gestion du contenu musical symbolique. Ce modèle est une extension du modèle relationnel classique incluant un type nouveau, le type *séquence temporelle*, que l'on utilise pour modéliser une voix monophonique. On introduit de plus la notion de *synchronisation des séquences temporelles* qui nous sert à modéliser une partition symbolique. On définit également un ensemble d'opérateurs algébriques pouvant agir sur ce modèle. Ces opérateurs peuvent se coupler à des fonctions utilisateurs, ce qui permet l'ouverture à des contextes applicatifs divers. On réussit donc à retrouver les opérations spécifiques de chaque domaine sans restreindre notre cadre de définition.

La description de l'algèbre servant à modéliser le contenu musical symbolique, c'est-à-dire le modèle choisi pour réprésenter le contenu musical et les opérateurs pouvant agir dessus, se trouve au chapitre 2. Notre modèle dépasse en réalité le cadre musical et peut s'étendre à n'importe quel contexte visant à manipuler, transformer et interroger des séquences temporelles, synchronisées ou non. Au travers d'exemples, nous proposons différentes applications possibles.

Le chapitre 3 introduit un langage utilisateur possible associé à cette algèbre, permettant d'exprimer des requêtes et de définir des fonctions utilisateurs.

Pour répondre à notre second objectif, nous adaptons un index existant, l'AS-index, aux données musicales. Cet index s'appuie sur les propriétés mathématiques de la signature algébrique d'un n-gram. Sa particularité est l'encodage d'informations musicales multiples pouvant intervenir dans la recherche. Grâce à cet encodage particulier, l'index permet de faire plusieurs types de recherche sur du contenu musical symbolique. Le même index autorise également la recherche approchée par motif. La description détaillée de l'index, ainsi que les définitions et propriétés des signatures algébriques, se trouvent au chapitre 4.

Le chapitre 1 est consacré aux travaux similaires ou reliés, et présente un état de l'art des différents domaines connexes.

Enfin, le chapitre 5 présente une application de ces travaux : une plateforme collaborative de ressources musicales symboliques.

Chapitre 1

Etat de l'art

Le sujet abordé dans cette thèse est à l'intersection de différents domaines de recherche. Il résulte que nous sommes amenés à faire intervenir ensemble des outils et concepts provenant de ces divers univers.

Dans ce chapitre, nous présentons un état de l'art des multiples domaines liés à notre sujet.

1 Séquences temporelles

Les séquences temporelles interviennent de manière privilégiée dans la construction de l'algèbre modélisant le contenu musical.

1.1 Généralités

Les séquences temporelles sont une forme de représentation de données qui apparait dans presque tous les domaines scientifiques et financiers. La définition la plus générale est la suivante.

Définition 2. *Une séquence temporelle est une séquence ordonnée d'événements observés à des dates successives.*

L'événement le plus simple est une valeur numérique. Une séquence temporelle modélise un phénomène éventuellement continu, mais la continuité ne pouvant être représentée on choisit une distribution dans le temps. Généralement, l'ensemble des dates est dénombrable et identifié à $\mathbb{N}$. Les événements sont séparés d'intervalles de temps égaux (des intervalles non égaux sont conceptuellement envisageables mais interdisent l'utilisation d'un grand nombre d'outils analytiques et statistiques).

Une séquence temporelle est donc une suite d'observations de la même variable. En pratique, tout ce qui est mesurable et varie en fonction du temps peut être représenté par une séquence temporelle.

1.2 Domaines d'application

Les séquences temporelles interviennent aujourd'hui dans des domaines trop divers pour pouvoir tous les citer, mais par exemple : en astronomie [48], en biologie [86], en théorie du signal [65], en météorologie [38], dans les domaines économiques

et financiers, pour des applications industrielles, et bien d'autres. Un historique de l'utilisation des séquences temporelles en économie est établi dans [16].

1.3 Méthodes d'analyse et utilisation des séquences temporelles

Les séquences temporelles intervenant dans des contextes applicatifs très divers, il est ambitieux de proposer une utilisation qui soit commune à tous. Citons entre autres l'analyse de tendances, la recherche par similarité, la recherche de motifs récurrents ou séquentiels [15].

Les études et analyses de séquences temporelles ont une vocation essentiellement statistique. Précisément, à partir de l'observation des données existantes, on établit des modèles pour pouvoir faire des prévisions (court terme) ou des prédictions (avenir lointain) [60]. L'analyse de ces séquences peut permettre de déterminer les états récurrents, stables ou attracteurs [39, 55]. On déduit également de ces observations les différents principes de causalités qui régissent le phénomène, ainsi que d'éventuels cycles récurrents. César et Richard dans leur cours [15] proposent plusieurs modèles prévisionnels :

- séquences déterministes : les événements futurs sont connus avec certitude ;
- séquences linéaires : croissance ou décroissance constante ;
- séquences auto régressives : chaque valeur dépend de la précédente.

Dans le cadre d'applications financières et scientifiques, le but essentiel de l'étude des séquences temporelles est la prévision (déterminer le futur en connaissant le passé) et le contrôle (suivant le contexte : maintenir ou au contraire éliminer les phénomènes récurrents). Dans ce contexte, le but de l'étude d'une séquence temporelle n'est pas de relier les variables entre elles, mais d'étudier la dynamique de l'ensemble, en répondant aux questions suivantes : la séquence est-elle stable, instable, stationnaire, saisonnière (un motif revient régulièrement), etc.

Dans le cadre d'une application commerciale, l'étude des séquences temporelles permet un meilleur contrôle du processus. Les cycles et tendances étant identifiés, on peut corriger les variations saisonnières. Les prévisions peuvent être intégrées en laissant une part dans le modèle aux phénomènes imprévisibles (ou bruit).

L'étude de la régression linéaire qui existe entre les différentes valeurs numériques de la séquence temporelle consiste à établir une relation affine liant ces différentes valeurs. Des généralisations de la régression linéaire sont exposées dans [16] : moyenne glissante (Moving Average MA), auto-régressive (AR) ou les deux (ARMA).

Les moyennes glissantes sont très utilisées en analyse de données boursières, leur fonction première étant d'aplanir les fluctuations de court terme pour donner la tendance générale de l'action.

1.4 Exemple : séquences temporelles et récupération d'information musicale

Les séquences temporelles offrent un cadre de travail adapté à la modélisation d'information évoluant avec le temps lorsque celui ci n'est pas un simple *time stamp* ou le calendrier classique. Elles apparaissent donc naturellement dans le domaine de

récupération d'information (*Information Retrieval*), et plus spécifiquement d'information musicale (*Music Information Retrieval*, ou MIR).

On retrouve dans le domaine du MIR de nombreuses problématiques importantes ayant trait aux séquences temporelles, notamment la recherche exacte ou approchée de motifs. Les méthodes utilisées traditionnellement pour l'analyse des séquences temporelles (modèles statistiques) sont par contre mal adaptées à l'analyse musicale.

Dans cet exemple, le domaine temporel est un temps abstrait représenté par une suite d'instants séparés d'intervalles égaux. Le domaine d'arrivée est un domaine complexe pour représenter l'information musicale (valeur des notes, rythme, texte, doigtés, nuances etc.).

La représentation de la musique par des séquences temporelles est une idée relativement répandue [2, 68, 49].

2 Bases de données temporelles

La modélisation de données évoluant dans le temps peut se faire par le biais de bases de données temporelles. Celles-ci sont en général destinées à représenter des données pouvant varier de manière imprévisible (comme le salaire d'un employé), contrairement aux séquences temporelles, plus intimement liées à la notion de motif temporel [51]. Bases de données temporelles et bases de séquences temporelles se distinguent aussi bien par leur fonctionnement que par leur utilisation. Il convient donc d'en exposer les principales caractéristiques afin de comprendre pourquoi nous les écartons.

2.1 Définitions

Une base de données temporelle est une base de données prenant en compte le facteur temporel des données. Le facteur temporel recouvre plusieurs notions de temps : le temps *valide*, le temps de *connaissance*, et le temps *transactionnel*.

Définition 3 (Temps valide)**.** *Le temps valide désigne la période de temps durant laquelle une donnée est vraie dans le monde réel (extérieur).*

Définition 4 (Temps de connaissance)**.** *On appelle temps de connaissance un instant ou un intervalle de temps associé à une donnée : suivant le cas soit le moment où la valeur de cette donnée devient connue, soit l'intervalle de temps où cette valeur est supposée être valide.*

Définition 5 (Temps transactionnel)**.** *Le temps transactionnel est la période de temps durant laquelle une donnée est stockée dans la base de données.*

Il est évident que *temps valide* et *temps transactionnel* ne coïncident pas nécessairement. En effet, une base de données modélisant des événements du XVIIIème siècle aura un temps valide allant des années 1701 à 1800, tandis que son temps transactionnel ne commencera qu'au XXème siècle, à la saisie des données.

Une base de données temporelle dispose donc d'un modèle de données intégrant cette notion de temps, ainsi que d'un langage adapté.

Les bases de données temporelles permettent d'interroger des données ayant une évolution au cours du temps, et qui ne doivent pas simplement être mises à jour ou supprimées, mais dont on veut conserver tous les états successifs. La description d'un objet peut en effet être fragmentée au cours du temps, et n'être complète que par l'énumération de ses états successifs.

Exemple 1. *Michel Legrand est employé depuis 20 ans dans la même société. S'il existe unee base de données temporelles des salaires des employés, on peut poser les questions « En quelle année Michel Legrand gagnait il moins de 2000 euros » et « Quel est le salaire de Michel Legrand en 2010 ? ».*

2.2 Historique, langage et implantations

Richard Snodgrass est à l'origine de nombreuses normes et conventions de la communauté des bases de données temporelles. Un glossaire des différents termes et concepts utilisés dans les bases de données temporelles est établi dans [41], ainsi que les discussions motivant les choix réalisés. L'ensemble des concepts sont rassemblés dans [77]. Les différents aspects théoriques de l'implantation d'un langage adapté ont été discutés en profondeur dans de nombreuses publications de Snodgrass[1].

TSQL2 est une extension *temporelle* du langage standard SQL-92. Il existe également des exemples de bases de données temporelles implantées dans des bases relationnelles : *Oracle Workspace Manager* par Oracle, *TimeDB* par TimeConsult.

2.3 Remarque

La différence fondamentale entre une base de données temporelles et une base de données de séquences temporelles réside dans le fait que d'un côté, on enregistre l'instant auquel a lieu un événement (changement d'état), et de l'autre on enregistre l'état à un instant donné.

3 Base de données de séquences temporelles

Les bases de données relationnelles ne sont pas conçues pour effectuer les opérations spécifiques aux séquences temporelles telles que des requêtes concernant des périodes précises ou des modifications du domaine temporel (conversion de fuseau horaire ou agrégation de 24 heures en un jour par exemple). La composition de ces opérations est encore plus difficile.

Une base de données temporelle n'est pas non plus nativement destinée à gérer des séquences temporelles. En effet, une séquence temporelle a deux aspects essentiels : l'intervalle de temps constant entre chaque événement, et le fait que la connaissance du passé est un indicateur du futur. Une base de données temporelle ignore ces deux aspects. L'événement enregistré à la position $i + 1$ est postérieur à l'événement i, mais il n'est pas possible de savoir à priori quel laps de temps les sépare, ni d'en deviner la nature après l'étude des i premiers [72].

Les applications conçues pour manipuler de grands volumes de séquences temporelles se voient donc obligées de développer spécifiquement certains outils.

[1] `http://www.cs.arizona.edu/~rts/publications.html`. Rubrique *Implementation of temporal databases*

3.1 Pré-requis d'une BDST

Une système de gestion de séquences temporelles doit respecter certaines contraintes :

- être capable de créer, manipuler, stocker et afficher ou imprimer les séquences temporelles ;
- combler intelligemment les valeurs absentes. Certains instants peuvent ne pas avoir de valeurs associées (par exemple le dimanche pour une séquence temporelle d'un indicateur boursier). Un système de gestion doit pouvoir substituer à ces valeurs absentes une valeur pertinente en fonction du contexte et de la séquence considérée ;
- préparer les données brutes, notamment les ajuster à l'échelle de temps choisie. Cette opération diffère suivant la nature de la variable observée. Un inventaire quotidien se transforme facilement en inventaire hebdomadaire en prenant simplement une valeur sur sept. En revanche, si la séquence représente les recettes quotidiennes, le passage à la recette hebdomadaire se fait par une somme. Réciproquement, convertir une séquence à indicateur hebdomadaire en une séquence à indicateur quotidien implique une opération d'interpolation (combler les valeurs manquantes). Cette interpolation appelle la même réflexion préalable que dans l'exemple précédent ;
- appliquer les opérations types des séquences temporelles, comme les requêtes agrégées (moyennes glissantes) ou les opérations de prévision (régression, corrélation, recherche de motifs).

3.2 Modèles et langages pour les séquences temporelles

Des modèles conçus pour les séquences temporelles sont proposés dans [20, 30, 70, 71, 69]. Dans [52] les auteurs définissent une algèbre proche de l'algèbre relationnelle ainsi qu'un langage de requêtes (SQL étendu) pour la manipulation de données ordonnées.

Une liste de références et rapports détaillés de nombreux modèles, ainsi que les langages de requêtes correspondants sont présentés dans [76, 18, 42].

Certains SGBD ont été adaptés pour gérer les séquences temporelles, mais ils reposent la plupart du temps sur des approches propriétaires, ce qui induit un coût important de développement. La majorité de ces systèmes est destinée à s'intégrer dans des applications financières, boursières ou de surveillance (données récupérées par des capteurs).

Le système FAME (*Forecasting, Analysis and Modeling Environment*[2]) permet de créer des séquences temporelles en spécifiant l'intervalle de temps, le type (*flow*, du type vente, ou *level*, du type stock), en les important depuis un fichier ou en les saisissant directement. Le système permet de générer de nouvelles séquences à partir des séquences pré-existantes par des opérations simples (calculer le profit à partir des ventes et dépenses), de faire des opérations du type interpolation, et des calculs prévisionnels en appliquant l'autorégression.

S-Plus[3] est un environnement d'analyse de données non spécifiquement destinées aux séquences temporelles [7]. De nombreuses fonctions mathématiques sont inclues

[2] `www.fame.com`

[3] `www.mathsoft.com/splus`

nativement, ainsi que des techniques de data mining pour la recherche. Le langage adapté est orienté objet, ce qui autorise l'encapsulation et la surcharge. Le système S-Plus ne permet pas de créer des séquences temporelles mais des vecteurs, sur lesquels on peut appliquer des opérateurs statistiques ou des opérations de type sélection. Pour compenser des performances médiocres, il est utilisé avec le système de base de données statistique SAS. Ce système est bien adapté aux séquences temporelles (la librairie ETS permet l'interpolation et intègre quelques outils d'analyse financière et de prévision). Le système permet de définir l'intervalle de temps d'une séquence, d'appliquer des calculs comme l'autorégression, des outils de traitements de données, et possède son propre SQL : Proc SQL

Le langage K est un langage propriétaire développé par Arthur Whithney et commercialisé par Kx Systems[4]. K est un langage conçu pour manipuler des tableaux, incluant aussi bien des fonctions opérant sur des tableaux en tant qu'entité, ou des opérations propres aux tableaux (tri ou inversion des éléments par exemple). K sert de base au système Kdb, une base de données orientée colonnes, ainsi qu'au langage d'interrogation correspondant KSQL (proche du SQL). Kdb est utilisé dans un grand nombre d'applications financières commercialisées par Kx Systems.

StreamBase Event Processing Platform[5] est un logiciel permettant de construire des systèmes qui analysent et agissent sur un flux de données en temps réel. Le langage utilisé est le StreamSQL, extension du SQL standard auquel s'ajoutent des opérateurs permettant de traiter des flux continus de données, de gérer des fenêtres de temps, ainsi que diverses fonctionalités du type fusion, agrégation, calcul etc.

Shasha introduit dans [73] la notion d'*arrable*, croisement de tableau (*array*) et table relationnelle, ainsi que le langage de requête correspondant AQuery. Le but est de permettre de faire des requêtes dépendantes de l'ordre, c'est-à-dire des requêtes dont le résultat change si l'ordre des lignes change (par exemple une requête faisant intervenir un calcul de moyenne glissante). Ces requêtes basées sur l'ordre peuvent être écrites en SQL mais de manière très fastidieuse, et sont quasiment impossibles à optimiser. Au travers d'exemples de requête, il établit la simplicité de la sémantique de AQuery relativement aux mêmes requêtes exprimées en SQL99.

3.3 Utilisation et analyse des BDST

Shasha [72] présente de manière détaillée une utilisation des séquences temporelles dans le milieu financier, circonscrivant avec précision les besoins des traders. Le but est de trouver dans un ensemble de séquences temporelles les séquences corrélées, au sens où l'évolution de l'une est connue en étudiant l'autre. L'intérêt de cette étude dans un cadre boursier est évident. Pour obtenir cette information, l'application qui gère les séquences temporelles doit pouvoir effectuer les opérations suivantes : des requêtes corrélées sur l'ensemble des séquences de la base, des requêtes corrélées sur des périodes données, et pouvoir pondérer ces requêtes suivant la période, l'histoire récente ayant plus d'importance que l'histoire ancienne.

Une base de données de séquences temporelles peut être interrogée analytiquement par le biais de la recherche de motifs récurrents. Cette approche permet de

[4] `www.kx.com`

[5] `www.streambase.com`

donner une vision synthétique des données de la base et de découvrir des règles d'association ou de classement. On peut citer [61, 57].

4 Bibliothèques numériques

Les bibliothèques numériques (ou *digital libraries* DL) sont des bibliothèques où les collections de documents sont stockées dans un format numérique par opposition au format classique (papier, microfilm...). Elles constituent un outil de recherche d'information.

La numérisation des documents (articles, livres) aussi bien que la constitution de collections de documents nativement digitaux répond aux mêmes besoins que les bibliothèques classiques : archivage, conservation, organisation, consultation. L'indexation complète du contenu donne naissance à de grandes possibilités de recherche ainsi qu'à de nouvelles problématiques (indexation pour une recherche robuste et performante). On assiste actuellement à des projets de numérisation des bibliothèques classiques de grande envergure (projet Google Library [10]).

De nombreuses communautés où interviennent recherche, organisation et commerce s'intéressent aux bibliothèques numériques, comme les universités (et universitaires), enseignants, maisons d'édition, scientifiques, historiens, conservateurs de musées.... Cet intérêt s'inscrit dans l'essor que connait le web 2.0 qui a pour fondement le partage des connaissances et la création de diverses communautés [67], encouragé par un accès quasi-illimité à des possibilités de stockage, bande passante et accès aux ressources pour les utilisateurs.

De nombreuses plateformes de connaissances (Wikipedia), images (Flickr), vidéos (Youtube), ou musique (iTunes) sont des exemples de cette utilisation nouvelle du Web et sont à la lisière du domaine des bibliothèques numériques.

4.1 Définitions et problématiques

Le terme de bibliothèque numérique admet deux définitions distinctes [12]. D'un côté, les chercheurs voient les bibliothèques numériques comme un contenu amassé pour une communauté d'utilisateurs, alors que les bibliothécaires voient les bibliothèques numériques comme des institutions ou services.

Différentes problématiques et défis animent la communauté des bibliothèques numériques :

- architecture technique : connexion réseau et internet, support de formats hétérogènes des documents, moteur de recherche plein texte pour l'accès aux sources, gestion générale des documents. Cette architecture doit de plus ne pas être figée mais au contraire être capable d'évoluer ;
- constitution des collections : numérisation, acquisition de documents digitaux, accès aux documents externes (liens). Cet aspect technique doit être complété d'une dimension théorique pour que la collection présente un intérêt : construction thématique d'une collection, rareté des documents, accès réservé à une communauté restreinte ;
- description des contenus : une description intelligente, homogène, pérenne pour rendre la recherche et la navigation possible et performante.
- copyright, propriété intellectuelle et gestion des droits ;

– préservation des documents numériques (disques...), de l'accès à l'information, de l'accès aux documents originaux (non numérisés).

4.2 Exemples

Le *Digital Libraries Initiative* (DLi)[6], projet lancé en 2005 par la Commission Européenne, a pour ambition de mettre en place la Biblitohèque Numérique Européenne, qui donnerait accès aux citoyens européens à l'héritage culturel et scientifique de chaque pays membre. Ce projet encourage entre autre chaque pays membre à investir dans la numérisation de ses œuvres et les rendre disponibles en ligne. Un autre effet secondaire de cette entreprise est l'approfondissement des questions de droits et de copyright. Les questions d'interopérabilité et de multilinguisme sont également à l'étude.

De nombreuses propositions d'annotations automatiques pour les bibliothèques numériques ont été faites. On peut citer le *Digital Library Annotation Service* (DILAS) [4], le *Collaboratory for Annotation Indexing and Retrieval of Digitized Historical Archive Material* (COLLATE) [43], et le *Flexible Annotation Service Tool* (FAST) [5]. L'annotation peut être manuelle ou automatique.

5 Rappel et précision

On mesure la performance d'un algorithme de recherche dans une base de données selon deux critères : le rappel et la précision. La précision est une mesure d'exactitude, le rappel est une mesure de complétude.

5.1 Rappel

Définition 6 (Rappel)**.** *Le rappel est le nombre de documents pertinents restitués divisé par le nombre total de documents pertinents existant dans la base de données.*

Lorsque un utilisateur interroge une base de données, il souhaite voir apparaître tous les documents répondant à son besoin d'information. Si cette adéquation entre le questionnement de l'utilisateur et le nombre de documents présentés est importante alors le taux de rappel est élevé. Si le taux de rappel est proche de zéro (c'est-à-dire si le système possède de nombreux documents intéressants mais que ceux-ci n'apparaissent pas en réponse) on parle de silence. Le silence s'oppose au rappel.

Définition 7 (Faux négatif)**.** *Un faux négatif est une réponse pertinente incorrectement écartée de l'ensemble des réponses à une requête donnée.*

5.2 Précision

Définition 8 (Précision)**.** *La précision est le nombre de documents pertinents restitués par la recherche divisé par le nombre total de documents restitués pour une requête donnée.*

[6] `http://ec.europa.eu/information\_society/activities/digital\_libraries`

Quand un utilisateur interroge une base de données, il souhaite que les documents proposées en réponse à son interrogation correspondent à son attente. Tous les documents retournés superflus ou non pertinents constituent du bruit. La précision s'oppose à ce bruit documentaire. Si elle est élevée, cela signifie que peu de documents inutiles sont proposés par le système et que ce dernier peut être considéré comme précis.

Définition 9 (Faux positif)**.** *Un faux positif est une réponse incorrectement considérée comme pertinente.*

5.3 Moyenne harmonique

Rappel et précision sont souvent inversement proportionnels, et l'on peut augmenter l'un en réduisant l'autre. En général la mesure de l'un est comparée à une mesure fixe de l'autre (par exemple mesure du rappel pour une précision de 0.75). On peut également les combiner dans une seule mesure appelée mesure-F qui constitue une moyenne harmonique des deux mesures.

Définition 10 (Mesure-F)**.**

$$2\frac{precision.rappel}{precision + rappel}$$

Dans cette définition, rappel et précision ont le même poids.

6 Notion de similarité

Une recherche par similarité, à distinguer d'une recherche exacte, retrouve les séquences de données qui diffèrent légèrement (ce critère étant à fixer par l'utilisateur) de la séquence de données passée en requête.

6.1 Généralités

Les principales problématiques liées à la recherche par similarité de séquences temporelles sont les suivantes :

- définir la similarité entre deux séquences temporelles (cette définition n'est pas nécessairement unique) ;
- calculer la similarité de deux séquences temporelles ;
- trouver de manière efficace deux séquences similaires (problème de l'indexation dans des requêtes par similarité).

Pour un ensemble donné de séquences temporelles, on peut distinguer deux types de recherche par similarité. D'un côté, trouver toutes les séquences temporelles similaires à une séquence donnée. De l'autre, trouver à l'intérieur d'un ensemble les séquences qui sont similaires les unes aux autres [59].

La première remarque que l'on peut faire est que la notion de similarité dépend du contexte applicatif, du type d'utilisateur et du but poursuivi. A chaque contexte correspond une ou plusieurs distances traduisant cette subjectivité.

On rappelle la définition d'une distance métrique.

Définition 11 (Distance). *On appelle distance sur un ensemble E une application de $E \times E$ dans $\mathbb{R}+$ vérifiant les propriétés suivantes :*
- *$d(A,B)=0 \Leftrightarrow A=B$ (séparation)*
- *$d(A,B)=d(B,A)$ (symétrie)*
- *$d(A,B) \geq 0$ (positivité)*
- *$d(A,C) \leq d(A,B) + d(B,C)$ (inégalité triangulaire)*

Lorsque l'inégalité triangulaire n'est pas vérifiée, on parle de *distance semi-métrique*. Par la suite, on désigne par distance à la fois les distances métriques et semi-métriques.

Une mesure de similarité quantifie la ressemblance entre deux séquences temporelles. On appelle parfois distance l'inverse d'une mesure de similarité (qui n'est pas nécessairement métrique dans ce cas).

Définition 12 (Recherche par similarité). *Etant donnée une séquence requête P, la recherche par similarité consiste à trouver toutes les séquences temporelles S d'une base Ω vérifiant*

$$\{S \in \Omega | d(P,S) < \varepsilon\},$$

où d est une distance et ε un seuil fixé par l'utilisateur.

6.2 Choix d'une distance

Le choix d'une distance pour mesurer la similarité entre séquences temporelles doit être suffisamment souple pour résister aux différentes transformations (dilatation de l'axe temporel, translation...) et sources d'erreur (présence de bruit, valeurs extrêmes non pertinentes). Il est indispensable de pouvoir comparer des séquences de longueurs différentes (recherche par motif ou *pattern matching*), ce qui exclut les distances classiques comme la distance Euclidienne ou L_p.

La dynamique d'ensemble de la séquence est dans certains cas plus intéressante que la succession des valeurs individuelles, pouvant contenir certaines anomalies. Dans ce cas un pré-traitement (du type moyenne mobile par exemple) des séquences permet d'en offrir une version plus synthétique. De nombreuses techniques de réduction de dimension des séquences temporelles existent et peuvent être choisies en fonction du contexte : *Piecewise Aggregate Approximation* (PAA), *Piecewise Linear Approximation* (PLA), transformation de Fourier discrète (DFT).

Une mesure de similarité permettant l'indexation présente un avantage supplémentaire.

6.3 Distance Euclidienne

La méthode la plus basique (naïve) pour une recherche par similarité est le calcul de la distance Euclidienne entre deux séquences. Ceci impose entre autre que les deux séquences soient de même longueur. Une méthode pour accélerer les recherches par similarité dans ce cas est proposée dans [36], et généralisée dans [64], où le cas de la recherche de sous-séquence est pris en compte (autorisant que la séquence recherchée soit plus courte que toutes les séquences de la base). On peut aussi adapter la distance Euclidienne afin d'autoriser quelques transformations sur les séquences, comme des translations ou des changements d'échelle [17].

Cette distance, quoique naturelle, est bien trop rigide pour les exemples de la vie réelle. La signification de la similarité varie énormément d'un contexte à l'autre. Deux séquences ayant la même forme peuvent être considérées comme similaires. Dans [63] on donne l'exemple de deux séquences ayant des valeurs ponctuelles differentes (distance Euclidienne supérieure à 11) mais dont les séquences représentant les moyennes glissantes sur trois jours sont elles très proches (distance Euclidienne inférieure à 0.5).

On peut trouver de nombreux exemples où la distance Euclidienne va considérer comme peu similaires des séquences temporelles que l'intuition – ou l'utilisateur – pourrait au contraire rapprocher [45]. Ceci vient du fait que la distance Euclidienne est très sensible aux petites distortions de l'axe temporel. Plus précisément, deux séquences ayant globalement la même forme doivent être considérées comme semblables si l'on peut correctement les aligner, c'est-à-dire obtenir des séquences équivalentes en déformant l'axe temporel.

6.4 Distance d'édition

La distance d'édition est la distance la plus utilisée pour mesurer la similarité entre deux chaînes de caractères [53]. Elle s'adapte très simplement aux séquences temporelles.

Définition 13 (Distance d'édition)**.** *La distance d'édition entre deux chaînes de caractères est le nombre d'opérations nécessaires pour transformer l'une en l'autre.*

Les opérations de transformations sont les insertions, suppressions et substitutions. Suivant les cas, certaines de ces opérations peuvent avoir un poids plus importants que d'autres. On trouve également des versions de cette distance qui incluent la transposition (interversion de deux caractères adjacents), erreur de saisie très fréquente.

Le pseudo-code 1 est un exemple d'algorithme de calcul de la distance d'édition.

Exemple 2. *Ci-dessous la matrice calculant la distance d'édition entre deux mots.*

		A	*V*	*I*	*R*	*O*	*N*
	0	*1*	*2*	*3*	*4*	*5*	*6*
A	*1*	**0**	*1*	*2*	*3*	*4*	*5*
V	*2*	*1*	**0**	*1*	*2*	*3*	*4*
I	*3*	*2*	*1*	**0**	**1**	*2*	*3*
O	*4*	*3*	*2*	*1*	*1*	**1**	*2*
N	*5*	*4*	*3*	*2*	*2*	*2*	**1**

Plusieurs variations existent dérivées de cette première définition.

Définition 14 (Distance de Hamming)**.** *La distance de Hamming entre deux chaînes de caractères de même longueur est le nombre de positions où les deux chaînes ont des caractères différents.*

Cette distance mesure le nombre de *substitutions* nécessaires pour transformer une chaîne en l'autre.

```
Input : P[1..m], Q[1..n]
Output : d_edit(P[1..m],Q[1..n])
d := array[0..m, 0..n];
int i, j;
for i = 0 to m do
 |  d[i, 0] := i
end
for j = 0 to n do
 |  d[0, j] := j
end
for j = 1 to n do
 |  for i = 1 to m do
 |  |  if P[i] = Q[j] then
 |  |  |  d[i, j] := d[i-1, j-1] // pas d'opération
 |  |  else
 |  |  |  d[i, j] := min(
 |  |  |  d[i-1, j] + 1, // suppression;
 |  |  |  d[i, j-1] + 1, // insertion;
 |  |  |  d[i-1, j-1] + 1 // substitution;
 |  |  |  )
 |  |  end
 |  end
end
return d[m,n];
```

Algorithme 1: Calcul de la distance d'édition entre deux séquences

6.5 Distance par déformation temporelle

La distance habituellement utilisée pour comparer deux séquences temporelles est la distance par déformation temporelle (*Dynamic Time Warping* ou DTW), une distance classique permettant un alignement non-linéaire entre deux séquences temporelles [3, 11].

La technique du DTW fait appel à une distance ponctuelle, dont le choix sera dicté par le contexte. Ceci permet de pondérer les erreurs. On peut par exemple décider que $d(x, y) = 1$ si $x \neq y$, pour tout x, y. Ce choix est pertinent si l'on cherche un motif dans un texte où peuvent exister des fautes de frappe (auquel cas aucune lettre erronée n'est plus fausse qu'une autre). Dans le cas où, au contraire, il existe une hiérarchie entre les fautes, on utilisera la distance Euclidienne.

On considère deux séquences temporelles $Q = q_1 q_2 \dots q_n$ et $P = p_1 p_2 \dots p_m$. Pour aligner ces deux séquences en utilisant le DTW, on créée une matrice $n \times m$ où le terme d'indice (i, j) est la distance $d(q_i, p_j)$ avec la distance qu'on s'est choisie.

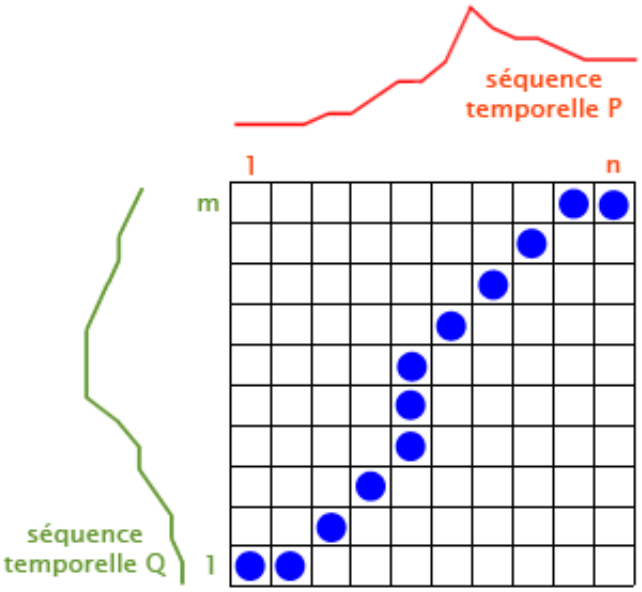


FIG. 1.1 – Chemin de déformation entre deux séquences temporelles

Un *chemin de déformation* $W = w_1 w_2 \dots w_K$ est un ensemble d'éléments contigus de la matrice, c'est-à-dire que si w_k, le k-ième élément de W, est l'élément d'indice (i, j) de la matrice, alors w_{k+1} est choisi parmi les trois possibilités suivantes : $(i + 1, j)$, $(i, j + 1)$ ou $(i + 1, j + 1)$.

La distance DTW(Q,P) est alors le choix de chemin qui minimise le coût d'alignement, autrement dit :

$$DTW(P, Q) = min \sqrt{\sum_{k=1}^{K} w_k}$$

Rappelons que chaque terme w_k de cette somme est une distance entre un élément de Q et un élément de P.

Le calcul de ce chemin minimisant peut être fait par récurrence. On définit la distance cumulée $\gamma(i,j)$ comme la distance $d(i,j)$ de la cellule courante additionnée à la plus petite des distances cumulées des cellules adjacentes

$$\gamma(i,j) = d(q_i, p_j) + min\{\gamma_{i-1,j-1}, \gamma_{i-1,j}, \gamma_{i,j-1}\}.$$

Cette technique permet entre autre de comparer des séquences de longueurs différentes.

On présente avec le pseudo-code 2 un exemple d'algorithme pour le calcul de la distance DTW.

Input : $Q[1..n]$, $P[1..m]$
Output : DTWDistance($Q[1..n]$, $P[1..m]$)
DTW= array[0..n, 0..m];
int i, j, *cost*;
for $j = 1$ *to* m **do**
  DTW[0, j] := infinity
end
for $i = 1$ *to* n **do**
  DTW[i, 0] := infinity
end
DTW[0, 0] := 0;
for $i = 1$ *to* n **do**
  for $j = 1$ *to* m **do**
    cost := $d(Q[i], P[j])$;
    DTW[i, j] :=
    cost + min(DTW[$i-1$, j], DTW[i , $j-1$], DTW[$i-1$, $j-1$]);
  end
end
return DTW[n, m];

Algorithme 2: Calcul de la distance DTW(P,Q)

Cette technique d'alignement est bien connue dans le domaine de reconnaissance du langage, qui a introduit le *Dynamic Time Warping* [66]. Il existe plusieurs algorithmes permettant d'accélerer les réponses tout en conservant la pertinence. De nombreux rafinements de cette technique ont été développés [46, 19].

Exemple 3. *Calcul de la distance DTW entre les séquences* $P =< 3, 6, 8, 7, 8 >$ *et* $Q =< 3, 5, 6, 5, 6, 8, 7 >$

Q/P	*3*	*6*	*8*	*7*	*8*
3	**0**	*3*	*8*	*12*	*17*
5	*2*	**1**	*4*	*6*	*9*
6	*5*	**1**	*3*	*4*	*6*
5	*7*	**2**	*4*	*5*	*7*
6	*10*	**2**	*4*	*5*	*7*
8	*15*	*4*	**2**	*3*	*3*
7	*19*	*5*	*3*	**2**	**3**

On trouve $DTW(P,Q) = 3$.

6.6 Longest Common Subsequence (LCSS)

La mesure LCSS (*Longest Common Subsequence*, ou *Plus longue sous-séquence commune*) est une méthode pour mesurer la similarité entre deux séquences utilisée à l'origine pour comparer des chaînes de caractères. L'idée est de saisir la notion intuitive que deux séquences peuvent être considérées comme similaires si elles ont suffisamment de sous-séquences similaires ne se chevauchant pas. Cette approche est adaptée pour une recherche en présence de bruit.

Définition 15 (Sous-séquence)**.** *Une sous-séquence est une suite d'éléments non nécessairement consécutifs d'une séquence (à la différence d'une sous-chaîne où les éléments sont consécutifs).*

Exemple : Si on considère les chaînes $ABCDEFGH$ et $IBJCKLMGNH$ la plus longue sous-séquence commune est $BCGH$. Les éléments sont dans le même ordre dans les deux séquences mais ne sont pas consécutifs.

Position du problème

Le problème LCSS consiste à trouver la plus longue séquence commune dans un ensemble de séquences (souvent réduit à deux). Ce problème est NP-difficile dans le cas général (nombre arbitraire de séquences) et peut se résoudre en temps polynomial dans le cas d'un nombre constant de séquences. La complexité peut encore être réduite suivant la taille des séquences, la taille de l'alphabet etc.

A noter qu'une LCSS n'est pas nécessairement unique. Leur longueur, en revanche, l'est. Lorsqu'on parle du problème LCSS, on cherche à exhiber explicitement *toutes* les plus longues sous-séquences. Dans le cas d'une mesure de similarité, on cherche seulement à établir leur longueur, ce qui est un problème plus simple.

Remarquons que le LCSS mesure une *similarité* et non une *distance*. Pour l'exprimer comme une distance, on a la formule

$$d_{LCSS}(A, B) = 1 - \frac{LCSS(A, B)}{\min(n, m)},$$

où A et$B$ sont deux séquences de tailles respectives n et m.

L'utilisation du terme *distance* est abusive car le premier axiome n'est pas vérifiée. Cette « distance » ne vérifie pas l'inégalité triangulaire.

LCSS métrique

On introduit à présent une autre définition de la distance LCSS qui cette fois ci est bien une distance métrique.

$$d_{LCSS}(A, B) = 1 - \frac{LCSS(A, B)}{\max(n, m)},$$

où A et$B$ sont deux séquences de tailles respectives n et m.

Cette distance devient un très bon candidat pour la recherche par similarité, puisqu'elle tolère les transformations d'axes temporels et vérifie l'inégalité triangulaire.

Calcul de la taille d'une LCSS

La fonction suivante prend deux chaînes $X[1..m]$ et $Y[1..n]$ et calcule la taille de la LCSS entre X et Y.

Input : $X[1..m]$, $Y[1..n]$
Output : LCSlength($X[1..m]$, $Y[1..n]$)
$C = \text{array}(0..m, 0..n)$;
for $i = 0$ *to* m **do**
  $C[i, 0] = 0$
end
for $j = 0$ *to* n **do**
  $C[0, j] = 0$
end
for $i = 1$ *to* m **do**
  for $j = 1$ *to* n **do**
    if $X[i] = Y[j]$ **then**
      $C[i, j] := C[i-1, j-1] + 1$;
    else
      $C[i, j] := \max(C[i, j-1], C[i-1, j])$;
    end
  end
end
return $C[m, n]$;

Algorithme 3: Calcul de la distance LCS

D'autres algorithmes sont exposés dans [33].

Avantages

Comme le DTW, le LCSS permet de comparer des séquences de tailles différentes.

La mesure LCSS est de plus moins perturbée par la présence de bruit que ne l'est la distance DTW. Les valeurs isolées ou extrèmes sont plus facilement ignorées. On peut autoriser les sauts dans la séquence.

LCSS$_\delta$

On adapte le principe du LCSS aux séquences temporelles en ne mettant en correspondance que les valeurs qui se ressemblent. Autrement dit on introduit une distance dans la mesure LCSS.

Précisément, pour deux séquences $A = (a_i)$ et $B = (b_j)$, a_i et b_j seront mis en correspondance si et seulement si $d(a_i, b_j) < \delta$. LCSS$_\delta$ calcule le nombre de valeurs mises en correspondance.

Cette adaptation de la distance LCSS aux séquences temporelles ne vérifie pas l'inégalité triangulaire et est donc semi-métrique [83].

Autres variantes

Dans [32], les auteurs proposent une mesure de similarité basée sur le LCSS pour des séquences temporelles de vecteurs de données hétérogènes en haute dimension (proche de notre modèle de séquences synchronisées). Les séquences temporelles

considérées modélisent des trajectoires qui synchronisent plusieurs paramètres hétérogènes. La mesure de similarité doit répondre à plusieurs contraintes, comme la présence de bruit, la translation et dilatation des séquences.

Une variante est proposée dans [59] : deux séquences temporelles sont considérées comme similaires si elles ont suffisamment de sous-séquences similaires ne se chevauchant pas. Deux sous-séquences sont considérées comme similaires si l'une est contenue dans une enveloppe autour de l'autre (de largeur définie par l'utilisateur). L'amplitude des deux séquences temporelles peut être reproportionnée de manière à rendre la recherche invariante par changement d'échelle.

6.7 Distances n-grams

On rappelle une définition.

Définition 16 (n-gram)**.** *Un n-gram est une sous-chaîne de n éléments consécutifs d'une chaîne donnée.*

Les n-gram sont utilisés en recherche approchée. En décomposant une séquence en une suite de n-gram, la séquence peut être représentée dans un espace vectoriel et donc comparée efficacement à d'autre séquences. L'inconvénient de cette technique est qu'une partie des informations de la séquence est perdue par ce découpage.

On peut remédier à ce défaut en introduisant dans la définition la position du n-gram. On obtient un n-gram *positionné* par glissement d'une fenêtre de longueur n le long d'une chaîne donnée. En début et fin de chaîne, les n-grams peuvent avoir moins de n éléments. On introduit alors de nouveaux caractères $\sharp$ et $\%$ n'appartenant pas à l'alphabet de départ et on ajoute $n-1$ $\sharp$ en préfixe de la chaîne et $n-1$ $\%$ en suffixe de la chaîne, créant ainsi une chaîne étendue dont tous les n-grams auront exactement n éléments

Définition 17 (n-gram positionné)**.** *Un n-gram positionné est un couple $(i, \sigma[i, \dots i+n-1])$ où $\sigma[i, \dots i+n-1]$ est le n-gram qui débute à la position i de la chaîne étendue.*

Deux chaînes sont considérées comme similaires si elles ont un grand nombre de n-grams communs de sorte que ces n-grams ne diffèrent que de quelques positions.

On peut choisir de découper une chaîne en n-grams qui se chevauchent (*overlapping*) ou non.

Définition 18 (Mesure de similarité cosinus)**.** *La similarité cosinus est une mesure de similarité qui calcule le cosinus de l'angle entre deux vecteurs. Cette mesure va de 1 (les deux vecteurs ont même sens et même direction, l'angle est nul) à 0 (les deux vecteurs sont orthogonaux).*

Dans [81] on donne la définition d'une distance basée sur le nombre de n-grams communs à deux séquences. Soient Σ un alphabet fini, $\Sigma^\star$ l'ensemble de toutes les chaînes d'éléments de Σ et Σ^n l'ensemble des chaînes de longueur n (n-grams) de Σ. Les n-grams peuvent être rangés suivant l'ordre lexicographique, on note donc l'ensemble $\Sigma^n = \{r_1, r_2, \dots, r_{|\Sigma^n|}\}$.

Définition 19 (Profil n-gram)**.** *Soit $x = a_1 a_2 \dots a_N$ une chaîne de $\Sigma^\star$ et soit r_j dans Σ^n un n-gram. Si $a_i a_{i+1} \dots a_{i+n-1} = r_j$ pour un certain i, alors x a une occurrence de r_j. Soit $G(x)[r_j]$ le nombre total d'occurences de r_j dans x. Le profil n-gram de x est le vecteur $G_n[x] = (G(x)[r_j])_{j \in [1, |\Sigma^n|]}$.*

Définition 20. *Soient x, y des chaînes de $\Sigma^\star$ et soit $n > 0$ un entier. La distance n-gram entre x et y est*

$$A_n(x, y) = \Sigma_{j=1}^{|\Sigma^n|} |G(x)[r_j] - G(y)[r_j]|.$$

A_n n'est pas strictement une distance puisque $A_n(x, y)$ peut valoir 0 même lorsque $x \neq y$.

On utilise cette distance entre autre pour son lien avec la distance d'édition. La distance d'édition et la distance A_n sont liées par l'inégalité suivante :

$$A_n(x, y)/(2n) \leq d_{edit}(x, y).$$

On se sert alors de la distance A_n comme *filtre* de la distance d'édition.

D'une manière très similaire, on a les définitions suivantes.

Définition 21 (Signature n-gram)**.** *La signature n-gram d'une séquence x est un vecteur $(c_i)_{i=1...|\Sigma|^n}$ où chaque coordonnée correspond à la présence ou l'absence du n-gram. Autrement dit, $c_i = 0$ si le n-gram r_i est absent de la séquence x, $c_i = 1$ sinon.*

Remarquons qu'on ne parle pas ici de l'histogramme des n-grams (profil n-gram) qui comptabilise le nombre exact de n-grams présents dans x.

Définition 22 (c-signature et distance)**.** *Soient $q = |\Sigma|^n$ et $sig^1(x) = (a_0, a_1, \ldots, a_{q-1})$ la n-gram signature d'une séquence x. On définit sa c-signature par $sig^c(u_0, u_1, \ldots, u_{k-1})$, où $k = [q/c]$ et $u_i = \Sigma_{j=ic}^{(i+1)c-1} a_j$. On définit la distance entre deux signatures $sig^c(x) = (u_0, \ldots, u_{k-1})$ et $sig^c(y) = (v_0, \ldots, v_{k-1})$ par*

$$SDist(sig^c(x), sig^c(y)) = \Sigma_{i=0}^{k-1} |u_i - v_i|.$$

On a ensuite le résultat suivant qui permet de filtrer la distance d'édition.

Lemme 1 (Filtre basé sur les c-signatures)**.** *Soient S et P deux séquences telles que $d_{edit}(P, S) \leq \epsilon$ et soient $sig^c(S) = (u_0, \ldots, u_{k-1})$ et $sig^c(P) = (u_0, \ldots, u_{k-1})$ leurs c-signatures respectives. Alors*

$$\Sigma_{i=0}^{k-1} |u_i - v_i| \leq 2n\epsilon.$$

Cette inégalité permet un meilleur filtrage que celle proposée dans [81]. On obtiendra donc moins de faux positifs.

6.8 Distances et mesures de similarité en modélisation musicale

Dans cette partie, nous proposons une rapide vue d'ensemble de quelques techniques spécifiquement développées pour la recherche approchée en contexte musical.

Dans [85] est proposé une méthode de recherche mélodique approchée appelée *Linear Alignment Matching* (LAM), basée sur une approche géométrique. Cette technique est appliquée à la recherche par chantonnement (*query by humming*), où les erreurs de tonalité et rythme sont fréquentes. La distance est calculée entre

des *vecteurs* notes, chaque vecteur étant composé de la hauteur et de la durée. L'algorithme permet également la prise en compte des insertions, suppressions et substitutions pour l'alignement.

Dans [78], deux mesures appelées *Earth Mover Distance* (EMD) et *Proportional Transportation Distance* (PTD) sont introduites pour mesurer la similarité mélodique. Les mélodies sont représentées par un ensemble de points pondérés. EMD mesure *l'effort* minimum nécessaire pour transformer un ensemble en un autre. Intuitivement, chaque point pondéré peut être vu comme un tas de terre, ou réciproquement comme un trou. EMD mesure ainsi le coût de transport de ces tas de terre pour remplir les trous correspondants. L'effort pour transporter un tas dans un trou de volume identique à une distance nulle est nul ; transporter un tas vers deux trous dont la somme des volumes est égal au volume du tas est un peu plus élevé, etc.

7 Recherche d'Information Musicale (MIR) et Bibliothèques Numériques de Partitions (DSL)

Le domaine de recherche d'informations musicales (*Music Information Retrieval*, ou MIR) est un domaine riche et pluridisciplinaire qui a émergé à la fin des années 90. Depuis dix ans, la conférence ISMIR (*International Society for Music Information Retrieval*) s'attache à fédérer les recherches et avancées du domaine selon ses axes dominants. On peut citer (sans être exhaustif) : encodage et représentation de la musique, recherche exacte et approchée de motifs, classification et identification automatique de pièces, bibliothèque numérique de grandes collections (archivage et recherche), tatouage, etc. Les acteurs du domaine sont nombreux : ingénieurs (traitement du signal), musicologues (représentation symbolique), analystes programmeurs (pattern matching). La palette des motivations est immense. Ce domaine très vaste conduit souvent à conjuguer des démarches scientifiques objectives et des notions plus subjectives [28, 50, 37].

Le nombre de sources musicales disponibles sur le net, l'hétérogénéité des contenus, leur complexité ainsi que la multitude des représentations possibles font du stockage de contenu musical un défi passionnant pour les communautés base de données et bibliothèques numériques. L'accès à ces sources nécessite des méthodes de stockage, d'interrogation et de parcours intuitives, efficaces et pérennes. On parle de bibliothèques de partitions numériques (*Digital Score Libraries* ou DSL).

7.1 Représentation de la musique et interrogation.

Plusieurs représentations sont possibles et sont par nature conceptuellement très différentes [13, 79]. Fac similé de partitions, MIDI (*time stamped events*), audio, métadonnées, contour, musique symbolique, notation musicale conventionnelle... A ces représentations diverses correspondent différentes techniques de recherche : reconnaissance optique de la musique (*Optical Music Recognition*), recherche par chantonnement (*Query By Huming*), recherche approchée par motif.

Nous laissons de côté la recherche audio ou par reconnaissance optique pour nous concentrer sur la recherche par contenu (représentation symbolique). Nous signalons juste que certains systèmes audio traduisent en fait le signal entré en une

description symbolique, ce qui rejoint notre domaine d'intérêt. La difficulté porte alors sur la segmentation du signal en notes distinctes. Cette difficulté est contournée par certaines plateformes de recherche par chantonnement (Musipedia[7]) en imposant que l'utilisateur chante des syllabes hachées.

La représentation de la musique monophonique par une chaîne de caractères est la plus intuitive, chaque caractère représentant une note, l'intervalle entre deux notes consécutives (pour une représentation indépendante de la tonalité, insensible à la transposition), ou la position d'une note par rapport à la précédente (plus aigue, plus grave, identique : c'est le codage du contour). De nombreuses techniques très connues de recherche textuelle peuvent ainsi être transposées à la recherche sur le contenu. La recherche exacte d'un motif dans une collection de partitions ne pose pas de problème majeur, c'est une simple recherche de sous-séquence faisant appel à des algorithmes de recherche de chaînes tels Boyer-Moore. En revanche, la recherche approchée est un défi plus important. La discussion sur la notion de similarité doit s'adapter aux spécificités de la problématique musicale.

7.2 Polyphonie

On distingue deux types de polyphonies [40] :
- polyphonie structurée divisée en un nombre fixe et constant de voix (*voiced*)
- polyphonie libre et non-structurée ne pouvant pas facilement se diviser en parties (*unvoiced* ou *voiced unspecified*)

La première catégorie comprend les pièces faisant intervenir plusieurs instruments monophoniques (musique de chambre d'instruments à cordes ou à vents), la seconde contient les partitions d'instruments polyphoniques (typiquement : piano, guitare).

Crawford et al. [26] dressent la liste des différents problèmes de recherche polyphonique et des techniques de *string matching* existantes pouvant les résoudre. Pour les polyphonies à voix distinctes : recherche exacte, recherche avec suppressions, recherche de motifs répétés, recherche d'un motif transformé (inversé...), recherche de motif distribué dans plusieurs voix, recherche d'accords et recherche approchée. Pour les polyphonies à voix non-spécifiées, on retrouve les problèmes de recherche exacte et de recherche de motif répété distribué sur plusieurs voix. La recherche approchée dans ce type de polyphonie est un problème ouvert très difficile. Cette étude propose toutefois différentes approches pour extraire des motifs de polyphonie non-structurée.

Pour classer ou effectuer des recherches dans des partitions monophoniques, [29] considère que la représentation relative d'une mélodie (*i.e.* la suite de ses intervalles) contient suffisamment d'information. On convertit dans un premier temps la suite de notes en une suite d'intervalles, qui est ensuite découpée en n-grams et indexée. L'indexation est totale et non tronquée arbitrairement en *incipit* ou en thèmes.

Ce modèle monophonique est repris pour la recherche ou classification polyphonique. [40] s'intéresse à la modélisation par des n-grams de la polyphonie du premier type, en particulier pour la classification des pièces. Le n-gram permet notamment de rendre compte de la probabilité d'un événement en fonction des événements qui le précèdent immédiatement. On extrait une des voix de l'ensemble sous forme d'une succession d'intervalles (représentation relative) et on la fragmente en n-gram. Une

[7] `http://www.musipedia.org/`

technique classique de classification monophonique est ensuite appliquée. On analyse a posteriori quelle voix est susceptible d'offrir les meilleures réponses.

Utiliser un modèle monophonique sur une pièce polyphonique du second type (*unvoiced*) nécessite d'appliquer au préalable un algorithme d'extraction de voix. On appelle *déclenchement* chaque instant où une ou plusieurs notes sont jouées dans la partition. La méthode *skyline* [80] prend la note la plus aigue de chaque déclenchement. Cette méthode est non-exhaustive mais permet de conserver un sens musical à la voix extraite.

L'approche proposée dans [27] procède en deux temps : tout d'abord les notes sont ordonnées par ordre de déclenchement (groupées lors de déclenchement simultané), ensuite les n-grams sont constitués en prenant toutes les combinaisons possibles de n événements à déclenchements successifs. Cette méthode est totalement exhaustive mais perd en sens musical. Pour effectuer une recherche, une séquence polyphonique est transformée suivant le même procédé et comparée à l'index par des méthodes classiques. On a donc une recherche polyphonique-dans-polyphonie qui utilise des principes monophoniques.

D'une manière générale, trouver des motifs monophoniques dans des pièces polyphoniques peut se faire en reprenant des méthodes existantes à supposer qu'on opère au préalable une séparation des voix [82] En revanche, le problème consistant à trouver des motifs polyphoniques est un problème ouvert et peu d'approches sont pour l'instant proposées.

Conklin [21] propose une approche verticale pour la recherche de motifs polyphoniques sur des pièces à la polyphonie structurée. Un pré-traitement est nécessaire sur les pièces pour les rendre homogènes verticalement : des notes sont artificiellement introduites sur chaque voix dès lors qu'un déclenchement à lieu dans l'une d'elles. On peut ensuite découper la pièce en tranches verticales. Ces nouveaux vecteurs peuvent être traités de diverses façons. La première est de calculer une valeur à partir de chaque vecteur, pour être ramené à un problème de type monophonique. Le défaut de cette méthode est le peu de souplesse par rapport à l'axe temporel. Pour remédier à ce point faible, Conklin [22] ajoute un indicateur de *continuation* permettant de signaler un événement dont le déclenchement a eu lieu précédemment. Pour une partition à deux voix, cet indicateur sera *st* lorsque les deux notes démarrent simultanément, *b-sw* lorsque la voix grave démarre alors que la voix aigue résonne, *t-sw* lorsque la voix aigue démarre alors la voix grave résonne. Cette technique, quoique s'appuyant sur un algorithme de recherche de motif monophonique, permet de rechercher des motifs polyphoniques de manière assez puissante.

7.3 Plateformes existantes

Il existe plusieurs plateformes musicales donnant accès à des bibliothèques numériques de tailles plus ou moins importantes. Ces plateformes proposent plusieurs types de services : des outils d'analyse musicale, d'éditions et annotation de partitions, ou une interface de recherche exacte ou approchée. Les applications développées s'appuient sur le format des données, musicXML étant le plus fréquent. MelodicMatch[8] propose en parallèle des outils d'analyse et un moteur de recherche

[8] `http://www.melodicmatch.com/`

par pattern ; on ne peut toutefois pas composer les deux. Wikifonia[9] est une plateforme type web communautaire, offrant la possibilité d'annoter les partitions. La recherche se fait sur des critères métadonnées liés aux pièces (auteur, titre, etc.) et non par contenu.

Les exemples de plateformes dédiées à la gestion de contenu musical sont nombreux, on peut citer par exemple, en plus de celles sus-mentionnées Mutopia[10] ou Musipedia[11]. Ces plateformes développent pour la plupart des services de stockage et recherche, en particulier des recherches par similarité. Quelques applications développent également des outils d'annotation ou d'édition. D'une manière générale, les services proposés s'appuient sur le format des données stockées, ce qui empêche de composer les différents services.

8 Indexation

L'indexation d'une mélodie est un problème central de la recherche d'information musicale. Rechercher un titre ou un auteur à partir d'une mélodie approximative est un problème courant. Suivant les cas, la mélodie peut être incomplète, transposée, subir des modifications rythmiques, être parasitée par du bruit, ou toute autre source d'erreur qui implique qu'une recherche approchée doit être conduite. Certaines techniques proposent une saisie vocale (mélodie sifflée ou chantonnée) mais traduisent ensuite cette mélodie en une version symbolique ou textuelle pour la comparer à la base de données. La recherche mélodique s'appuyant sur la programmation dynamique (calcul de la distance d'édition) fonctionne mais passe mal à l'échelle. En effet, le calcul de la distance d'édition est quadratique en la taille de la requête, et implique un parcours séquentiel de la base. De plus, de par sa nature la distance d'édition ne peut être indexée par un simple index inversé [9]. On cherche alors à adapter des techniques d'indexation développées dans d'autres domaines pour obtenir un sur-ensemble du résultat, qui sera ensuite filtré par la distance d'édition.

8.1 Indexation de séquences temporelles

Lorsque l'utilisation des séquences temporelles est intensive, le volume de données mises en causes devient rapidement gigantesque et réclame une gestion efficace. On peut alors faire appel à des techniques de compression [84] ou l'indexation des séquences temporelles [47].

Faloutsos [64] liste les propriétés désirables pour une bonne méthode d'indexation :

- plus rapide qu'un parcours séquentiel ;
- permet les requêtes de tailles différentes ;
- permet les insertions et suppressions sans avoir à reconstruire l'index ;
- ne crée pas de faux négatifs ;
- renvoie peu de faux positifs.

[9] `http://wikifonia.org/`
[10] `http://www.mutopiaproject.org`
[11] `http://www.musipedia.org`

L'indexation des séquences temporelles se fait surtout en utilisant des structures d'indexation multidimensionnelle, ou des index de chaînes de caractères. Les techniques standards d'indexation de texte ne s'adaptent pas parfaitement au cas des séquences temporelles pour plusieurs raisons. L'absence de notion équivalente à celle du mot rend l'indexation des séquences temporelles difficile et ce problème est donc largement étudié. Une autre difficulté capitale est la haute dimension des données en jeu.

On distingue deux types d'indexation : l'indexation *exacte* qui retourne la meilleure réponse (qui serait celle renvoyée après un parcours séquentiel de la base) ; et l'indexation *approchée* qui renvoie des bons candidats, mais pas nécessairement le meilleur [47].

Lorsqu'une distance vérifie l'inégalité triangulaire, elle est plus facile à indexer [8]. Comme il a déjà été dit, une recherche par similarité utilisant le DTW est bien plus adaptée aux séquences temporelles que celle utilisant la distance Euclidienne, en revanche, celle-ci est moins simple à indexer car elle n'obéit pas à l'inégalité triangulaire. De nombreux chercheurs ont ainsi porté leurs efforts sur une indexation approchée (qui suffit dans beaucoup d'applications), ou se sont concentrés sur l'accélération du parcours séquentiel. [47] propose une méthode d'indexation exacte des distances basée sur une technique de borne inférieure.

Les techniques classiques d'indexation approchée consistent en une réduction de la dimension des séquences temporelles, suivie de l'indexation de cette donnée réduite. Plusieurs techniques de réduction des dimensions ont été proposées (une liste dans [74]), des techniques mathématiques classiques adaptées aux séquences temporelles (SVD, DFT, DWT) choisissant une représentation commune pour toutes les séquences qui minimisent l'erreur de reconstruction, ou des techniques spécialement développées pour les séquences telles que ACPA ou PAA.

- *Singular Value Decomposition* (SVD) ou décomposition en valeurs singulières : tirée d'un procédé d'algèbre linéaire pour la factorisation des matrices rectangulaires, permet de filtrer les données en fonction de leur importance relative.
- *Discrete Fourier Transform* (DFT) ou transformée de Fourier discrète : énormément utilisée en théorie du signal pour la compression des données, la DFT donne une représentation spectrale discrète d'un signal discret. Appliquée aux séquences temporelles, elle permet de décorréler les données de départ et de ne travailler que sur un petit nombre de coefficients significatifs.
- *Piecewise Aggregate Approximation* (PAA) : technique introduite par Keogh, on découpe une séquence temporelle en segments de longueur égale et on prend une valeur moyenne pour chaque segment. Cette technique extrêmement simple se révèle étonnamment compétitive.
- *Adaptive Piecewise Constant Approximation* (ACPA) : [44] chaque séquence est approchée par la valeur moyenne d'un segment de taille variable, de manière à minimiser l'erreur de reconstruction pour chaque séquence. La différence avec toutes les techniques précédentes est que la représentation n'est pas commune à toutes les séquences de la base, la longueur des segments découpant la séquence variant d'un cas à l'autre.

Les efforts les plus récents ont porté sur le passage à l'échelle de l'indexation, étant donné que les bases de données de séquences temporelles dépassent de plusieurs ordres de grandeur les tailles de bases de données classiquement étudiées

dans la litérature. La représentation SAX et son amélioration iSAX permettent une indexation des séquences temporelles pour une recherche exacte rapide et une recherche approchée ultra rapide dans de très grands volumes de données [74]. Les séquences temporelles sont réécrites en mots d'un alphabet de dimension inférieure de sorte que les séquences temporelles semblables seront représentées par le même mot. Chaque requête est traduite dans le même alphabet et rapatrie l'ensemble des séquences temporelles ayant la même écriture (recherche approchée). Pour la recherche exacte on combine l'algorithme de recherche approchée avec des distances ayant une borne inférieure. Cette technique évite les faux négatifs. Il est intéressant de noter que dans ce cas on n'utilise plus le DTW mais à nouveau la distance Euclidienne. Ce choix est motivé par l'observation que dans les grandes bases les séquences temporelles n'ont plus besoin d'être alignées.

8.2 Alignements

On parle d'*alignement* lorsque l'on cherche des régions à forte similarité entre deux longues séquences. Issus originellement de la bioinformatique, les alignements de séquences sont utilisés dans d'autre domaines où les objets manipulés sont comparables à des séquences ADN.

On distingue deux types d'alignements, les alignements globaux et locaux. L'alignement global force un alignement sur toute la longueur des séquences, quitte à insérer des sauts (ou trous) dans les séquences. Au contraire, l'alignement local identifie des régions similaires dans des séquences pouvant être par ailleurs très différentes. L'alignement local cherche la similarité *maximale* entre deux sous-séquences. Si les séquences sont augmentées, la similarité décroit.

Le calcul de l'alignement local se fait soit par programmation dynamique (long mais exhaustif), soit par heuristique (efficace mais non exhaustif). L'algorithme Smith-Waterman [75] est un exemple d'algorithme d'alignement local par programmation dynamique.

Les méthodes à base de recherche de *mots* sont plus efficaces que la programmation dynamique mais ne garantissent pas de trouver l'alignement optimal. Ces méthodes consistent à identifier de courts segments dans la séquence requête qui sont ensuite mis en correspondance avec des séquences de la base de données. La position relative des mots dans les séquences sont soustraites pour obtenir un offset : s'il existe plusieurs mots distincts qui produisent le même offset on a trouvé une région où l'alignement sera bon. On étudie ensuite de manière plus exhaustive les régions trouvées. Ceci permet d'élaguer de grandes portions de séquences où ce test plus fin est inutile.

BLAST [6] et FASTA [62] sont deux outils très populaires chez les biologistes. Les deux techniques utilisent le même principe : recherche de *mots* ou *germe* (*seed*) de longueur fixe, puis alignement des segments trouvés pour obtenir la meilleure approximation.

BLAST recherche les correspondances exactes de segments de bases de taille ω contigus, qui sont ensuite étendus à gauche et à droite pour obtenir l'alignement final. Le problème de cette méthode est qu'augmenter la valeur ω diminue la sensibilité de l'algorithme (la sensibilité mesure la probabilité qu'un bon alignement soit trouvé par l'algorithme), et réduire la valeur de ω augmente le nombre de réponses non-

pertinentes.

Ces deux méthodes font partie des indexations s'appuyant sur un filtrage, c'est-à-dire écartant de grandes régions de séquence où il n'existe pas de réponse à la requête.

8.3 n-grams

Cao et al. [14] proposent une indexation à deux niveaux (table de hachage et arbre) permettant également un filtrage pour l'indexation de la distance d'édition. On s'appuie entre autres sur l'observation que deux séquences contiennent un nombre important de n-grams communs si la distance d'édition est inférieure à un certain seuil. Cette méthode reprend en les améliorant les techniques développées par Ukkonen dans [81].

Le problème de la recherche approchée est posé de la manière suivante :

Problème 1 : Pour une longueur l et une distance d'édition τ, trouver toutes les sous-séquences S telles que $|S| \geq l$ et $d_{edit}(S, Q') \leq \tau$ pour une sous-séquence Q' de la séquence requête Q.

Ce problème est lui même réduit au problème suivant :

Problème 2 : Pour une longueur ω et une distance d'édition ε, trouver tous les segments s_i de taille ω tels que $\mathrm{d}_{edit}(s_i, q_j) \leq \varepsilon$ pour tout segment q_j de longueur ω inclus dans la requête Q.

Chaque séquence est découpée en *segments* de taille fixe ω. On génère des clusters de n-grams similaires (nClusters). Le premier niveau d'indexation est la construction d'une table de hachage sur les segments respectivement aux nClusters. Dans le second niveau d'indexation, les segments sont transformés en leur c-signature, et indexés par un arbre de c-signature.

9 Conclusion et positionnement

En introduisant un contenu aussi peu conventionnel que peut l'être le contenu musical symbolique dans l'univers des bases de données et des bibliothèques numériques, nous espérons faire progresser en parallèle aussi bien le domaine de la recherche d'information musicale que les possibilités offertes par les plateformes numériques.

En effet, devant gérer les caractéristiques de ce contenu inhabituel, les bibliothèques numériques rencontrent et surmontent de nouveaux défis ; de même la souplesse apportée par une approche base de données permet de faire progresser les outils classiques de recherche d'information musicale.

Par ailleurs, les idées naissant de l'étude du contenu musical dépassent largement ce domaine. Si l'on fait abstraction de la dimension musicale du contenu, notre approche nous permet d'aborder un problème plus général.

A notre connaissance, il n'existe pas de modèle représentant le contenu de données synchronisées suffisamment souple pour être utilisable dans plusieurs domaines. Si les séquence temporelles vectorielles sont déjà connues, elles ne sont pas utilisées comme nous le ferons, c'est-à-dire comme un ensemble de voix synchronisées pouvant être prises séparément ou au contraire superposées pour obtenir de nouvelles séquences. En effet, ces opérations nécessitent d'avoir été définies rigoureusement

pour être menées sans danger. C'est ce qui motive l'introduction d'une algèbre opérant de manière fermée sur ces objets. Les opérateurs de projection (permettant d'isoler une voix) et de produit (pour la synchronisation de plusieurs voix) sont un exemple d'opérations possibles.

Les bases de données temporelles ou les bases de données de séries temporelles sont encore trop proches des domaines d'application pour lesquelles elles ont été développées, mettant en avant des fonctionalités, certes avancées et performantes, mais rigides et non pérennes. Notre approche consiste à étendre le modèle relationnel en incluant un nouveau type, les séquences temporelles, mais également un certain nombre d'opérateurs nouveaux qui leurs sont destinés.

Les plateformes musicales existantes mettent en avant un nombre fini de fonctionnalités, portant sur un type de contenu bien déterminé. Nous nous distinguons de celles ci en cherchant à mettre en place une plateforme affranchie des contraintes imposées par le contenu qu'elle est sensée acceuillir. La plateforme doit être conçue de manière à pouvoir stocker, gérer et étudier des collections entières donc la structure n'est pas connue a priori, sans pour autant en modifier l'architecture. Celle-ci peut sans cesse s'enrichir de nouveaux outils, au gré de l'apparition des collections et des nouveaux besoins.

L'indexation de la recherche exacte est indispensable si l'on considère les volumes de données en jeu dès que des séquences temporelles interviennent. Par ailleurs, une contrainte supplémentaire tient à la nature de ces données et aux multiples façons de les interroger. Il n'existe pour l'instant pas de structure d'index unique permettant de réaliser plusieurs types d'interrogation. Pour cela, nous adaptons un index dont le principe est proche des index n-grams. Cet index permet également de conduire des recherches approchées par motif.

Le point de départ de ce travail est donc la gestion du contenu musical symbolique. Mais les questions soulevées par ce sujet permettent d'envisager des problèmes plus généraux. Le modèle unifié et l'approche base de données que nous proposons peuvent être repris et adaptés à de nombreux types de données synchronisées. De même, il est intéressant de voir dans quelle mesure différents types d'interrogations peuvent avoir un sens dans un domaine autre que musical.

Chapitre 2

Modélisation des séquences temporelles et application aux sources musicales symboliques.

Le travaux exposés dans ce chapitre ont fait l'objet de deux publications : *A Database Approach to Symbolic Music Content Management* [34] et *Modeling Synchronized Time Series* [35].

Dans ce chapitre, nous proposons un modèle logique spécialement destiné à la gestion des *séquences temporelles*. On s'intéresse en particulier à la notion de synchronicité de séquences temporelles qui conceptualise l'alignement de plusieurs séquences partageant un même domaine temporel.

Le modèle que nous proposons est une extension du modèle relationnel incluant le type *séquence temporelle*, ainsi que plusieurs opérateurs rassemblés dans deux algèbres agissant sur ce modèle, notamment une algèbre temporelle permettant d'opérer sur le temps. Les opérateurs de l'algèbre peuvent être combinés à des fonctions définies par un utilisateur.

Notre modèle repose sur le concept de séquences temporelles synchronisées. Intuitivement, une séquence temporelle synchronisée est un groupe de séquences temporelles partageant un domaine temporel. Cette définition fournit un cadre de travail pertinent pour l'étude de séquences temporelles devant être alignées, comparées ou fusionnées.

Dans ce chapitre, nous décrivons de manière formelle le modèle comme une extension du modèle relationnel, et nous décrivons précisément les opérateurs de l'algèbre qui opère de manière fermée sur ce modèle.

Nous montrons que (i) cette algèbre permet de retrouver toutes les opérations usuelles sur les séquences temporelles, (ii) une haute expressivité est atteinte grâce à la composition non bornée des opérateurs, et (iii) la possibilité de pouvoir introduire des fonctions utilisateur dans les expressions permet de ne pas limiter l'utilisation du modèle à un domaine applicatif spécifique.

Cette approche pose les bases pour la conception d'un système dédié à l'étude des séquences temporelles à grande échelle.

1 Motivation

Un système se bornant à stocker et interroger les séquences temporelles est trop limité dans un contexte applicatif, et il est nécessaire de pouvoir y combiner des fonctions d'analyse de manière fluide. A notre connaissance, les outils permettant de réaliser ces opérations restent spécialisés et s'appuient trop fortement sur le format des données, ce qui rend le couplage de ces opérations laborieux, voire impossible.

Par ailleurs, un système conçu pour gérer des séquences temporelles doit pouvoir faire appel à des outils analytiques et statistiques s'appliquant de manière globale ou locale. Ceci impose de pouvoir intervenir sur le domaine temporel. En conséquence, les séquences temporelles ne sont pas un cas particulier de données temporelles et ne peuvent pas être traitées facilement dans un système de gestion de base de données temporelles.

Les séquences temporelles peuvent être à valeur scalaire ou vectorielle. Les séquences vectorielles sont des séquences synchronisées partageant un domaine temporel commun. On dit alors que la séquence temporelle est un ensemble de *voix* synchronisées, chaque voix prenant ses valeurs dans des domaines a priori différents. La synchronisation de séquences temporelles peut avoir plusieurs fonctions : soit la simultanéité d'événements de natures différentes (par exemple les différents indices boursiers de plusieurs valeurs, le texte associé à une note dans la partition d'un chanteur), soit la juxtaposition de plusieurs ensembles de données à comparer (comme la consommation électrique mensuelle d'un particulier).

Les séquences temporelles permettent donc de représenter deux informations cruciales : la chronologie (succession des événements) et la synchronisation (simultanéité des événements). Autrement dit, les lectures verticales et horizontales sont possibles et pertinentes.

2 Exemples de séquences temporelles

Les séquences temporelles interviennent dans de nombreux domaines, et, même si notre exemple privilégié reste le contenu musical symbolique, le modèle présenté ci-après ne se limite pas à cet exemple précis. Dans cette partie nous présentons différents exemples d'applications où les séquences temporelles peuvent apparaitre, ainsi que les besoins utilisateurs associés.

2.1 Applications industrielles

Les séquences temporelles produites dans des contextes industriels sont liées à la production et/ou la consommation de produits ou de services. On considère par exemple un fournisseur d'électricité qui surveille la consommation quotidienne de ses clients sur une année. On peut représenter les séquences temporelles *elec_quot* de la manière suivante.

Le domaine temporel $\mathcal{T}$ est une suite de jours de 0 à 364, appelé *Calendrier*. Le domaine **dom** est l'ensemble des réels positifs $\mathbb{R}+$.

Une telle séquence temporelle est susceptible de subir les opérations suivantes :

Consommation maximale sur une période donnée. Choisir une période restreinte du calendrier et déterminer les consommations minimales ou maximale sur cette période.

Moyenne glissante. Déterminer la consommation hebdomadaire moyenne pour tous les jours de l'année. Remarquons que la réponse de cette requête est à nouveau une séquence temporelle.

Séquence temporelle agrégée. Chaque dimanche, donner la consommation cumulée de la semaine.

Produire et comparer de nouvelles séquences temporelles. Extraire les douze séquences de consommation mensuelle de la séquence temporelle donnant la consommation annuelle. Superposer ces séquences et calculer divers indices statistiques.

Dans plusieurs de ces exemples, on modifie le domaine temporel. Dans le dernier exemple, on transforme le simple *Calendrier* en un domaine plus abstrait des *jours du mois*.

La superposition des séquences temporelles est un exemple de synchronisation.

2.2 Trajectoires

On considère à présent une compagnie de livraison possédant plusieurs véhicules. Chaque jour, les trajets de chaque véhicule sont déterminés en fonction des lieux devant être visités. Ces trajectoires partagent un domaine temporel commun : les heures de la journée en cours. On peut donc les modéliser comme des séquences temporelles synchronisées qui, à chaque heure du jour, associent leurs positions respectives sur un plan.

Les requêtes sur ce type de séquences temporelles sont de différentes natures. Par exemple :

Projection/sélection : extraire le trajet d'un véhicule particulier à une période spécifique de la journée.

Jointure : montrer les véhicules qui sont dans la même région au même moment.

Plus proche voisin : étant donné un véhicule, trouver à chaque instant les véhicules les plus proches de lui.

D'autres opérations sur ces séquences (non spécifiques à l'objet en mouvement), telles que les moyennes glissantes ou calculs statistiques sur une période donnée, ont déjà été mentionnées et s'envisagent de la même façon.

Ce second exemple est similaire par beaucoup d'aspects au premier. Seuls les domaines temporels diffèrent, ainsi que les opérations liées aux domaine d'application. L'exemple suivant est plus particulier.

2.3 Modélisation musicale

Pour notre dernier exemple, on considère une bibliothèque de partitions numériques. Une partition consiste en plusieurs voix qui sont chacune modélisées par une séquence temporelle. La figure 2.1 est un exemple de pièce monophonique, et la

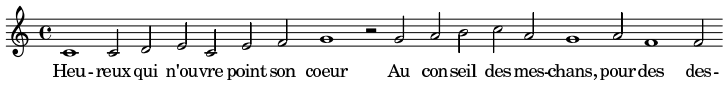


FIG. 2.1 – Partition monophonique

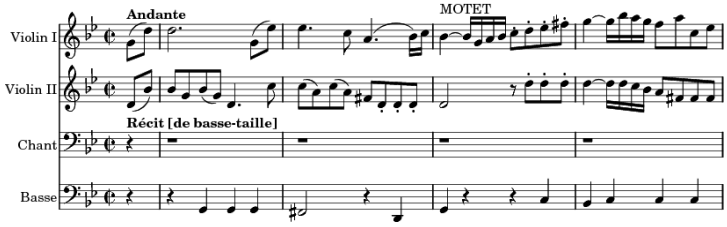


FIG. 2.2 – Partition polyphonique

figure 2.2, pièce polyphonique, donne un exemple de synchronisation de plusieurs voix.

Le domaine temporel n'est plus le *Calendrier*, mais devient une représentation abstraite du temps, qui précise dans quel ordre sont jouées les notes, quelles notes apparaissent ou sonnent simultanément, et permet de connaitre la durée de chaque note.

Des requêtes plus spécifiques au domaine musical sont alors imaginables.

Trouver toutes les partitions dont les paroles contiennent le mot « conseil » (sélection) ;

Trouver le fragment mélodique correspondant au mot « conseil » (sélection et jointure temporelle) ;

Trouver un fragment mélodique (recherche par similarité).

3 Présentation du modèle

3.1 Préliminaires

Notre modèle est une extension du modèle relationnel auquel on ajoute un type nouveau, le type séquence temporelle. L'algèbre relationnelle est étendue à ce nouveau type, et des opérateurs spécialement dédiés aux séquences temporelles sont regroupés dans une algèbre nouvelle : l'algèbre temporelle.

3.2 Notations et définitions

On considère un domaine de valeurs $\mathcal{D} = (\mathbf{dom}, \mathcal{O}, \Gamma)$, où **dom** est un ensemble de *valeurs*, $\mathcal{O}$ est un ensemble de *fonctions utilisateurs* sur **dom** et Γ est un ensemble de *fonctions d'agrégation*.

Définition 23 (Fonctions utilisateur, fonctions d'agrégation)**.** *Soit* ***dom*** *un ensemble de valeurs.*

Un opérateur op d'arité k dans $\mathcal{O}$ *envoie un k-tuple* $[v_1, \ldots, v_k]$ *de* $\mathbf{dom}^k$ *vers* ***dom****.*

Une fonction d'agrégation γ *de* Γ *envoie une suite de valeurs de* ***dom****,* $< v_1, \ldots, v_n >$*, dans* ***dom****.*

On considère que l'ensemble **dom** contient toujours deux valeurs privilégiées : le *neutre* $\top$ et le *nul* $\bot$.

De même, il existe toujours deux opérateurs booléens $\wedge$ (conjonction) et $\vee$ (disjonction).

La conjonction et la disjonction vérifient, pour chaque $a \in \mathbf{dom}$, $a \wedge \bot = \bot$, $a \wedge \top = a$ et $a \vee \bot = a \vee \top = a$ ($\vee$ et $\wedge$ sont commutatifs). Sauf précision contraire, pour chaque $op \in \mathcal{O}$, op est étendu par $\wedge$ quand au moins un opérande est $\bot$ ou $\top$, *i.e.* pour chaque $a \in \mathbf{dom}$, $op(\bot, a) = \bot$, $op(\top, a) = a$.

Le domaine de valeurs $\mathcal{D}$ ainsi que les fonctions d'agrégation et opérations dépendent du contexte d'application. Quand il n'y a pas d'ambiguité, on désigne $\mathcal{D}$ par « le domaine ».

Définition 24 (Domaine temporel, fonctions temporelles)**.** *On appelle domaine temporel* $\mathcal{T}$ *un ensemble dénombrable ordonné isomorphe à* $\mathbb{N}$*.*

On appelle fonction temporelle une fonction de $\mathcal{T}$ *dans* $\mathcal{T}$*. On note* $\mathcal{L}$ *la classe des fonctions temporelles.*

Une sous-classe importante de $\mathcal{L}$ est l'ensemble des fonctions linéaires $t \mapsto \alpha t + \beta$. On les appelle *fonctions de changement d'échelle*, et on distingue les familles d'homothéties ($warp^\alpha : \mathcal{T} \to \mathcal{T}$, $t \mapsto \alpha t$) et de translations ($shift^\beta : \mathcal{T} \to \mathcal{T}$, $t \mapsto t + \beta$).

Les éléments de $\mathcal{L}$ se composent quand cela est possible avec le classique opérateur mathématique $\circ$.

Définition 25 (Séquences temporelles)**.** *Soit* $\mathcal{D}$ *un domaine de valeurs de* ***dom*** *et* $\mathcal{T}$ *un domaine temporel. Une* séquence temporelle *s sur* $\mathcal{D}$ *est une fonction de* $\mathcal{T}$ *dans* ***dom****.* (TS) *désigne l'ensemble des séquences temporelles.*

On suppose le lecteur familier avec les concepts classiques du modèle relationnel et on choisit de ne pas les rappeler ici.

On désigne par $(\mathcal{R})$ l'ensemble des noms de relations, $(\mathcal{A})$ l'ensemble des noms d'attributs, et $(\mathcal{S})$ l'ensemble des noms de séquences temporelles.

Définition 26 (Schéma d'une (TS)-relation)**.** *Soit* R *le nom d'une relation,* $A_1, \cdots, A_k$ *des noms d'attributs de* $(\mathcal{A})$*, et* $S_1, \cdots, S_p$ *des noms de séquences temporelles de* $(\mathcal{S})$*. Alors* $R(A_1, \cdots, A_k, S_1, \cdots, S_p)$ *est le schéma d'une relation.*

Une *instance* de R est un ensemble fini de *tuples* $(a_1, a_2, \cdots, a_k, s_1, \cdots, s_p)$, où chaque s_i est une instance de S (*i.e.*, une séquence temporelle), et $a_1, \ldots, a_k$ sont des instances d'attributs relationnels classiques.

3.3 Exemples

Nous illustrons à présent par des exemples les notions introduites dans la partie précédente.

Domaines simples

Une séquence temporelle peut être à valeurs dans un domaine de type classique (entiers, flottants...) qui sont alors désignés par $\mathbf{dom}_{type}$ ou plus simplement explicitement par leur type (exemple $\mathbf{dom}_{float}$, ou plus simplement *float*).

Un domaine classique peut toutefois avoir un nom plus représentatif de ce qu'il désigne. Par exemple pour modéliser une mélodie on utilise $\mathbf{dom}_{pitch}$, qui est un sous ensemble de *int*. De même, $\mathbf{dom}_{lyrics}$, ensemble de syllabes, peut être identifié à *string*.

Domaines complexes

Les domaines complexes (produits de domaines) ont l'intérêt supplémentaire qu'ils permettent de représenter la synchronicité. Pour représenter du contenu musical, on peut choisir le domaine des notes, accords, syllabes, doigtés, ou tout autre indication (tempo, nuance).

Le domaine **vocal** est un domaine produit

$$\mathbf{dom}_{vocals} = \mathbf{dom}_{pitch} \times \mathbf{dom}_{rythm} \times \mathbf{dom}_{lyrics}.$$

Pour ne pas alourdir les notations, les domaines unités (*rythm, lyrics, pitch*) sont parfois simplement notés *rythm, lyrics, pitch*.

Le domaine **polymusic** représentant la musique polyphonique est le domaine produit

$$\mathbf{dom}_{polymusic} = (\mathbf{dom}_{pitch} \times \mathbf{dom}_{rythm})^N.$$

Exemples de schémas

On note **TS([dom])** le type de domaine de valeurs d'une séquence temporelle.

Consommation électrique. La consommation électrique d'un groupe de particuliers sera modélisé par :

Energy (Id : **int**, Client : **string**, Consommation : **TS(float)**).

Trajectoires. Une société de livraisons employant une flotte de camionnettes peut se servir du schéma :

Routes (Id : **int** , Vehicule : **string**, Position : **TS(point)**)

Ici **point**=*float* x *float*, représente les coordonnées sur une carte.

Partitions. Pour gérer une bibliothèque de partitions, on choisira le schéma suivant :

Score (Id : **int**, Compositeur : **string**, Voix : **TS(vocal)**, Piano : **TS(polyMusic)**).

Remarquons qu'on a choisit de séparer les parties de piano et de chant, mais le domaine temporel reste implicitement partagé.

3.4 L'algèbre relationnelle ALG_R

Trois opérateurs relationnels classiques (sélection σ, projection π, produit cartésien $\times$) sont étendus au type séquences temporelles. Pour illustrer, on prend comme exemple le schéma *Score* de la section précédente.

Sélection

La sélection σ sur un attribut classique est bien connue.

$$\sigma_{compositeur='Louis\ Couperin'}(Score)$$

En revanche, la sélection sur un attribut de type séquences temporelles peut porter sur un fragment de la voix

$$\sigma_{vocals \to lyrics \supset 'Heureux\ Seigneur'}(Score)$$

On cherche un motif **contenu** dans une voix.

Projection

De même, on étend l'opérateur de projection afin de pouvoir projeter sur une partie du domaine **dom** :

$$\pi_{vocals \to lyrics}(Score).$$

Produit

Enfin, le produit cartésien étendu aux séquences temporelles permet de synchroniser des voix partageant le même domaine temporel. On peut le voir comme l'opération inverse de la projection, modulo quelques précautions sur le domaine temporel.

Si l'on considère par exemple une collection de duos séparés en partitions alto et soprano, avec les schémas suivants

$$Alto(Id : \textbf{int}, Voice : \textbf{TS(vocals)}),$$

$$Soprano(Id : \textbf{int}, Voice : \textbf{TS(vocals)}).$$

Pour obtenir les partitions des duos, on fait le produit cartésien entre deux partitions de tessiture différente. On obtient la relation

$$Duet(Id : \textbf{int}, Duo : \textbf{TS(vocals)}).$$

Jointure

En soi, le produit cartésien n'a qu'un intérêt limité, mais associé à la sélection il devient l'opérateur de jointure. Dans l'exemple précédent, on n'associe pas aléatoirement deux parties, mais seulement celles partageant le même identifiant.

$$\sigma_{A.Id=S.Id}(Alto \times Soprano) \equiv Alto \bowtie_{Id=Id} Soprano.$$

Test vide (*emptyness*)

L'équivalent du *null* pour les séquences temporelles est la séquence vide (*empty*), c'est-à-dire la séquence qui à chaque instant associe l'événement $\perp$. On introduit alors un dernier opérateur relationnel, le test *emptyness* $\emptyset^?$ tel que pour une séquence temporelle s, $\emptyset^?$(s)=true si et seulement si $\forall t, s(t) = \perp$.

Cet opérateur s'emploie associé à un opérateur de sélection.

3.5 L'algèbre temporelle $\text{ALG}_{(TS)}$

On introduit à présent les opérateurs de l'algèbre temporelle qui sont spécifiquement conçus pour manipuler des séquences temporelles.

L'algèbre temporelle $\text{ALG}_{(TS)}$ est composée de trois opérateurs $(\circ, \oplus, \mathbf{A})$. Chaque expression de $\text{ALG}_{(TS)}$ prend une ou plusieurs séquences temporelles en entrée et produit une séquence temporelle. Les opérateurs sont : la compositon externe $\circ$, l'addition $\oplus$, et un mécanisme **A** combinant une dérivation à une fonction d'agrégation de Γ.

Composition

Définition 27 (Composition)**.** *Soit s une séquence temporelle et $l \in \mathcal{L}$. Alors $s \circ l$ est la séquence temporelle définie par*

$$[s \circ l](t) = s(l(t)).$$

Cet opérateur permet de transformer le domaine temporel et ainsi de cibler un sous-ensemble d'événements. On dit qu'on génère une vue locale de la séquence.

Exemple 4. *La fonction $shift^n$ est un élément de la famille de fonctions shift, paramétré par une constante $n \in \mathbb{N}$. Pour toute séquence temporelle $s \in (TS)$ et à chaque instant $t \in \mathcal{T}$, $s \circ shift^n(t) = s(t+n)$. Autrement dit, $s \circ shift^n$ est la séquence obtenue à partir de s d'où on a retiré les n premiers événements.*

Addition

L'opérateur d'addition permet de propager au niveau de la séquence temporelle une fonction opérant sur des valeurs atomiques.

Définition 28 (Addition)**.** *Soient $s_1, s_2, \ldots s_p \in (TS)$ et $op \in \mathcal{O}$ une fonction d'arité p. Alors $\oplus_{op}(s_1, s_2, \ldots, s_p)$ est la séquence temporelle définie par*

$$\oplus_{op}(s_1, s_2, \ldots, s_p)(t) = op(s_1(t), s_2(t), \ldots, s_p(t)).$$

Remarque : Dans la pratique, on ne s'intéresse qu'aux opérateurs d'arité 1 ou 2.

Exemple 5. *On considère deux séquences temporelles* alto *et* soprano *représentant deux parties vocales. Les voix alto et soprano sont des voix à valeurs dans* **vocal**, *soit* **int** $x\ \Sigma^\star$.

On veut obtenir la séquence temporelle s de la progression harmonique entre ces deux voix.

On définit une fonction utilisateur harm *prenant deux notes en entrée et produisant l'intervalle qui les sépare de la manière suivante :*

$$\begin{array}{rccccc} harm: & vocal & \times & vocal & \rightarrow & interval \\ & (n, \_) & \times & (m, \_) & \mapsto & |n-m| \end{array}$$

La séquence temporelle de la progression harmonique entre les deux voix est alors définie de la manière suivante :

$$s = \oplus_{harm}(alto, soprano).$$

Remarquons que dans l'exemple précédent, on a généré une séquence temporelle dont le domaine est différent de celui des séquences passées en entrée.

Dérivation

Nous nous tournons à présent vers le mécanisme de dérivation-agrégation. Les opérateurs introduits jusqu'à présent agissent sur des valeurs discrètes, et ne peuvent pas déduire de valeurs à partir de sous-séquence (extrait de séquences). Une opération permettant par exemple de calculer une moyenne sur une fenêtre glissante est pour le moment absente de notre algèbre.

Pour combler cette lacune, nous nous inspirons du principe de group by / agrégation de l'algèbre relationnelle classique. A partir d'une séquence temporelle s, on dérive une **famille** de séquences temporelles $s' =< s'_1, \ldots, s'_n >$ de la façon suivante : en chaque instant i on déduit une séquence temporelle s'_i de s en la composant avec une fonction temporelle. Autrement dit, pour tout i, il existe une fonction temporelle λ telle que

$$s'_i = s \circ \lambda.$$

On applique ensuite une fonction d'agrégation sur chaque série de cette **famille**.

Cet opérateur procède donc en deux temps :

1. premièrement, à chaque instant τ, une séquence temporelle s'_τ est dérivée (déduite) de s grâce à un *opérateur de dérivation* d_λ où $\lambda \in \mathcal{L}$. On dit que l'on dérive s suivant λ en τ.
2. deuxièmement, une *fonction d'agrégation* $\gamma \in \Gamma$ est appliquée sur chacun des s'_τ, produisant un élément du domaine **dom** en chaque instant τ.

La figure 2.3 montre une version schématisée de ce double mécanisme. On donne à présent les définitions.

Définition 29 (Dérivation)**.** *Soit $\lambda = \{\lambda^i, i \in \mathbb{N}\}$ une famille de fonctions temporelles et s une séquence temporelle. La* dérivation *de s suivant λ est une fonction de $(TS) \rightarrow (TS)^{\mathbb{N}}$:*

$$d_\lambda(s) = (s \circ \lambda^0, s \circ \lambda^1 \ldots, s \circ \lambda^n, \cdots).$$

Le plus souvent on utilise la famille de fonctions temporelles

$$shift = (shift^0, shift^1, \ldots, shift^n)$$

qui permet de prendre la famille des sous-séquences commençant aux instants successifs.

On combine l'opérateur de dérivation avec une fonction d'agrégation du domaine, et on obtient l'opérateur **A** de $\text{ALG}_{(TS)}$.

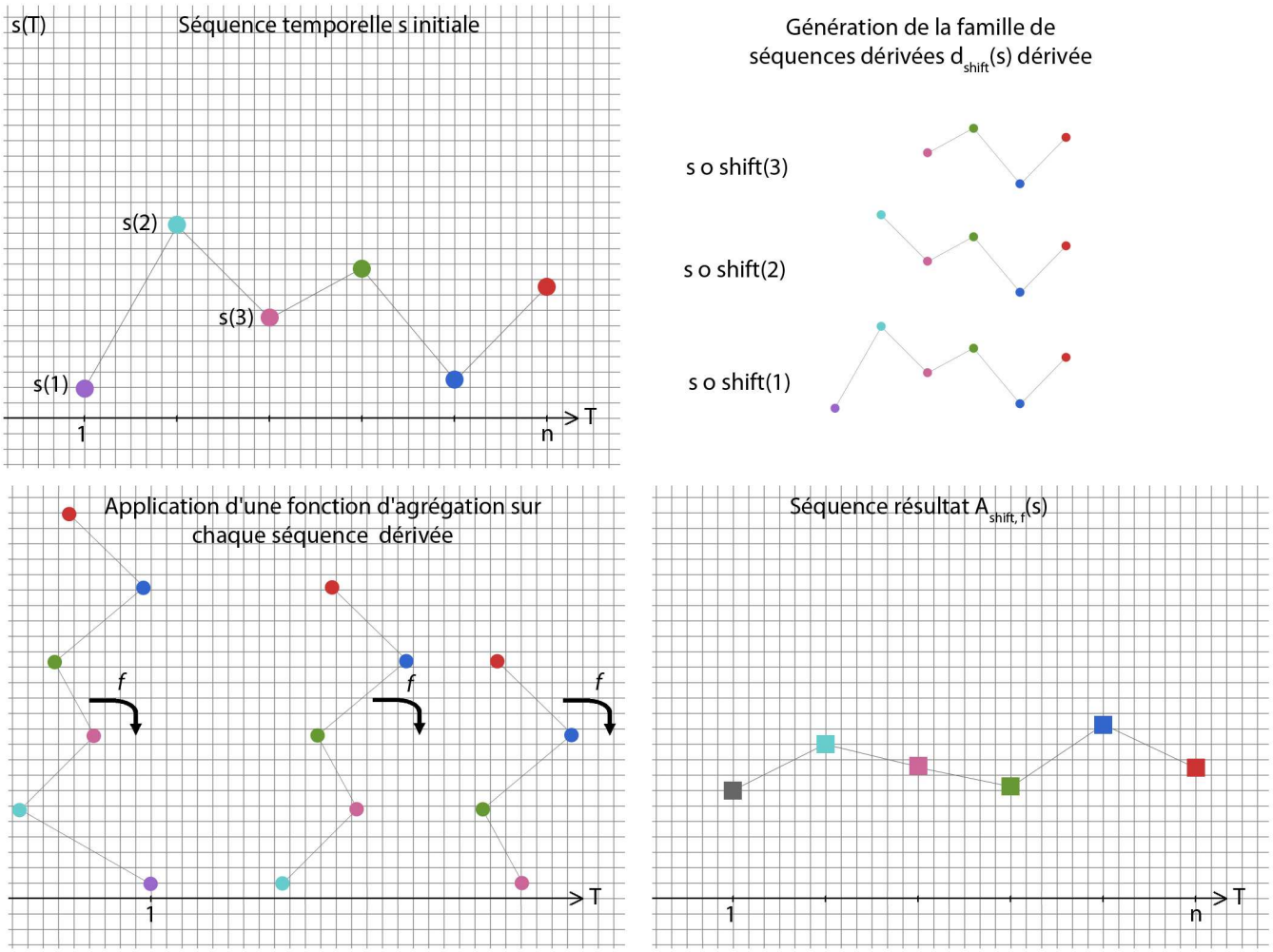


FIG. 2.3 – Dérivation - Agrégation

Définition 30 (Agrégation). *Soit γ une fonction d'agrégation, et λ une fonction temporelle. Alors, pour toute séquence temporelle s, $\boldsymbol{A}_{[\gamma,\lambda]}(\mathrm{s})$ est une expression de l'algèbre $\mathrm{ALG}_{(TS)}$ définissant la séquence temporelle :*

$$[\boldsymbol{A}_{[\gamma,\lambda]}(\mathrm{s})](\mathrm{t}) = \gamma(\mathrm{s} \circ \lambda^{\mathrm{t}}).$$

3.6 Exemple

L'exemple suivant illustre la définition précédente. On fait une recherche approchée d'un motif à l'intérieur d'une voix. Cette recherche passe par un calcul de distance entre un motif P et une séquence temporelle s. La distance habituellement utilisée pour comparer deux séquences temporelles est le DTW (Dynamic Time Warping), une distance classique permettant un alignement non-linéaire entre deux séquences temporelles ([3, 11]).

Exemple 6. *On cherche une approximation du motif P au sein d'une séquence temporelle s. On applique la dérivation-agrégation à s respectivement en dérivant selon la famille de fonctions temporelles* shift*, et en agrégeant avec la fonction $dtw_P = dtw(., P)$ qui calcule la distance entre P et une séquence temporelle donnée. Alors*

$$\boldsymbol{A}_{\mathrm{dtw_P},\mathit{shift}}(\mathrm{s})$$

est la séquence temporelle qui à chaque instant n donne la distance entre P et la séquence temporelle extraite de s $s \circ \mathit{shift}^n$.

Un opérateur de sélection est ensuite appliqué à ce calcul de distance pour ne garder que les séquences contenant des motifs dont la distance à P est inférieure à une distance maximale.

4 Conclusion

Le modèle présenté permet d'inclure les séquences temporelles de manière générique et haut niveau dans un système de gestion de base de données. « Haut niveau » signifie ici que la représentation des données et l'ensemble des opérations associées ne gênent pas la conception d'une représentation physique efficace, et permet un stockage physique complètement indépendant. « Générique » concerne l'aspect abstrait du modèle logique qui le rend indépendant du contexte applicatif.

En adoptant dès le début une approche algébrique, on rend possible la définition d'un langage expressif et stable, ne nécessitant pas une redéfinition au cas par cas. Compte tenu de la grande diversité des domaines manipulant des séquences temporelles, un aspect important de notre travail est la possibilité d'introduire nativement dans les expressions des fonctions définies par un utilisateur.

Chapitre 3

Langage de requêtes

Le modèle unifié présenté au chapitre 2 permet de conduire des requêtes sur des partitions venant de collections différentes, c'est-à-dire n'ayant pas le même schéma de données. De simples requêtes sur les attributs classiques aussi bien que des requêtes plus complexes impliquant des manipulations de contenu peuvent être menées.

Dans ce chapitre, nous présentons un langage de requête correspondant au modèle défini dans le chapitre précédent. Notre modèle étant très proche du modèle relationnel, seules quelques extensions du SQL classique sont nécessaires pour obtenir un langage utilisateur adapté.

Le langage associé au modèle doit avoir une sémantique précise pour garantir la non-ambiguité des requêtes. Il doit spécifier *ce qui doit être fait* et non *comment faire*.

En plus des requêtes, on veut pouvoir définir des fonctions utilisateurs, elles mêmes appelables au sein des requêtes.

Idéalement, la syntaxe est suffisamment éloquente pour que la lecture d'une requête permette d'en comprendre le sens.

1 Préliminaires

Pour donner au lecteur un avant-goût des possibilités du langage, on introduit dans cette partie le langage de manière discursive.

Une requête standard a la structure suivante :

from	tables
let	variable := expression
construct	expressions
where	attribut = valeur

Une telle requête signifie qu'on veut aller chercher dans les tables définies par la clause **from** les lignes respectants les critères introduits par la clause **where** et formatées de la manière précisée par la clause **construct**.

La clause **let** permet d'introduire des variables intermédiaires, pouvant intervenir à la fois dans la clause **where** et dans la clause **construct** (l'une, l'autre, ou les deux à la fois).

Par souci d'efficacité, les calculs inhérents à la clause **let** peuvent n'être effectués que s'ils sont nécessaires (par exemple lorsque les variables définies par **let** interviennent dans **construct** mais pas dans **where**).

On donne à présent de manière informelle les quelques mots clés permettant la manipulation des séquences temporelles.

map/map2 : applique une fonction utilisateur à chaque valeur d'une séquence temporelle.

contains : extension de la sélection aux séquences temporelles.

empty : test de vide sur une séquence temporelle, renvoie true si la séquence est vide.

derive(_,_,_) : opérateur de dérivation-agrégation, prend en arguments la séquence temporelle à dériver, la famille de fonctions temporelles et la fonction d'agrégation.

synch : synchronise deux séquences temporelles partageant un domaine temporel commun.

comp : compose une séquence temporelle avec une fonction temporelle (manipulation du domaine temporel).

Les opérateurs de l'algèbre introduite au chapitre 2 peuvent tous être exprimés grâce à ces mots clés.

2 Syntaxe

2.1 Structure d'une requête

Il y a quatre clauses principales : **from**, **let**, **construct**, **where**.

Dans toute la suite, on désigne par *attribut* à la fois les attributs classiques et les séquences temporelles.

La clause **from** énumère une liste (non vide) de tables de la base de données, avec un alias éventuel. Les noms de tables réfèrent à de vraies tables de la base de données. Les alias ne doivent pas être dupliqués, ni être des noms de tables existantes.

La clause **let** est optionnelle, et il peut y avoir autant de clauses **let** que souhaité. Dans une clause **let**, on applique une fonction utilisateur à un attribut. Les calculs portent sur des attributs présents dans les tables listées par la clause **from**. Quand l'attribut est une séquence temporelle, on utilise le mot-clé **map** pour appliquer la fonction utilisateur à la séquence. Si on applique un opérateur binaire, on utilise **map2**.

La clause **where** est optionnelle. Elle permet de spécifier le prédicat filtrant les résultats affichés par **construct**. Ce prédicat est un ensemble de tests (égalité, comparaison, contains...) reliés par des opérateurs logiques (And, Or, Not). Les conditions de la clause **where** sont évaluées sur chaque ligne. Ces conditions portent sur des attributs introduits par **let** ou **from**. Si la condition est évaluée comme vraie, alors la ligne à laquelle elle réfère fait partie du résultat. Une absence de clause **where** équivaut à écrire **where true** .

La clause **where** supporte les opérateurs arithmétiques (+, −, ∗, /), booléens (And, Or, Not) et de comparaison (=,<>, >,<,>=, <=, contains), qui apparaissent à l'intérieur des prédicats.

Les attributs apparaissant dans la clause **construct** peuvent provenir soit des tables listées par la clause **from**, soit d'une clause **let**. La clause **construct** donne la liste des attributs (définis par un **let** ou non) qui doivent apparaitre dans la réponse de la requête.

Les attributs qui apparaissent dans les clauses **construct**, **let** et **where** sont désignés par leur nom (`ColumnName`) lorsqu'il n'y a pas d'ambiguité (c'est-à-dire quand le nom de la colonne apparait dans une seule des tables listées), ou en précisant leur table d'origine dans le cas contraire (`TableName.ColumnName`).

Pour accéder à une voix d'une séquence temporelle (projection sur une voix), on utilise la syntaxe `->`. Précisément, une voix est appelée `ColumnName->VoiceName` ou `TableName.ColumnName-> VoiceName`.

2.2 Grammaire du langage

Nous décrivons ici formellement ce que l'on peut écrire dans le langage sous la forme de règles de grammaires.

Une commande entrée par l'utilisateur correspond à la règle `<instruction>`, c'est-à-dire une requête (`<from>`) ou la définition d'une fonction (`<define>`). La règle `<main>` représente une série de commandes.

```
<main> ::= <instruction>*

<instruction> ::= <from> | <define>

<from> ::=
 from <named_var_list> <let>* construct <expr_list> (where <expr>)? ;

<define> ::= define <var> ( <var_list> ) = <expr> ;

<var_list> ::= <var> | <var> , <var_list>

<named_var_list> ::= <named_var> | <named_var> , <named_var_list>

<named_var> ::= <var> <var>?

<let> ::= let <var> := <expr>

<voice> ::= <var> - > <var> | <var>.<var> - > <var>

<expr_list> ::= <expr> | <expr> , <expr_list>

<expr> ::=
         | <expr> <binop> <expr>
         | <unop> <expr>
         | <var>
         | <var>.<var>
```

```
          | <fun> (<var_list>)
          | <voice>
          | <literal>

<binop> ::=
          | + | * | ...
          | = | < | > | ...
          | contains
          | and
          | or

<unop> ::= not | -        | is null | is not null

<fun> ::= <var> | map | map2 | derive | ...

<var> ::= [a-zA-Z$_][a-zA-Z0-9_]*

<literal> ::= <int> | <float> | <string> | ...

<int> ::= (+|-)?[0-9]+

<float> ::= (+|-)?[0-9]*.[0-9]+

<string> ::= "(a-zA-Z)*"
```

3 Traduction algébrique du langage

Toute requête de notre langage peut s'exprimer en une expression algébrique. Nous décrivons ici formellement comment traduire une requête en une expression algébrique. Les opérateurs algébriques ayant été définis formellement au chapitre 2, cette relation nous donne alors la sémantique du langage.

Remarque. *La notation algébrique des opérateurs* **map**, **derive** *et* **comp** *n'est pas la même que dans le langage* ($\oplus$, $\boldsymbol{A}$, $\circ$)*. Pour simplifier les règles de traduction, on considère qu'on les écrit de la même manière.*

D'après la grammaire, une requête peut s'écrire
`from T* let b* construct e*`
ou
`from T* let b* construct e* where P`
où `T*` désigne un ensemble de tables, `b*` un ensemble éventuellement vide d'associations variables/expressions, `e*` un ensemble d'expression, et `P` un prédicat.

On se ramène à un seul cas en considérant que la première forme est équivalente à `from T* let b* construct e* where true`.

Pour exprimer simplement la traduction, on se munit de l'opérateur `[ ]` qui permet de remplacer une variable par une expression dans une expression. Plus formellement `e[x↦e']` signifie qu'on remplace toutes les occurences de `x` par `e'` dans `e`. Pour chaque règle de grammaire définissant `<expr>`, on définit le remplacement de

la manière suivante (`x`, `y` et `z` sont des variables, `l` est un litéral, `b` est une association variable/expression) :

`x[x↦e]`	`e`
`x[y↦e]`	`x`
`l[b]`	`l`
`(e <op> e')[b]`	`e[b] <op> e'[b]`
`(x.y)[b]`	`(x[b]).(y[b])`
`(x->y)[b]`	`(x[b])->(y[b])`
`(x.y->z)[b]`	`(x[b]).(y[n])->(z[b])`
`(not e)[b]`	`not (e[n])`
`(f(v1, v2, ...))[b]`	`f(v1[b], v2[b], ...)`

De la même manière et avec la même notation, on définit le remplacement successif de plusieurs variables par leurs expressions associées.

Cet opérateur nous permet de définir facilement la traduction d'une requête en son expression algébrique :

```
from T* let b* construct e* where P
```

se traduit par

$$\Pi_{e*[b*]}\sigma_{P[e*]}(T*)$$

4 Exemples

Dans cette partie, nous illustrons grâce à de nombreux exemples la traduction des opérateurs algébriques introduits au chapitre 2 en leurs équivalents syntaxiques.

4.1 Algèbre relationnelle

Projection : Le projection se note algébriquement :

$$\Pi_A(R),$$

où R est une relation, A est un ensemble d'attributs de R. Le résultat est la relation R où l'on ne considère que les attributs de A. Son équivalent syntaxique est la clause **construct**.

Sélection : La sélection se note algébriquement :

$$\sigma_F(R),$$

où F est une formule, R est une relation. Le résultat est

$$\{r \in R \mid r \text{ satisfait la condition donnée par } F\}.$$

Son équivalent syntaxique est la clause **where** $\ldots = \ldots$.

Exemple : l'expression

$$\Pi_{id,voice}(\sigma_{composer='Faure'}(\textbf{Psalms}))$$

correspond à la requête

from	Psalms
construct	id, voice
where	compositeur='Faure'

Remarque : Ici *voice* est une séquence temporelle.

Extension aux séquences temporelles :

Projection : La projection sur une séquence temporelle se note algébriquement :

$$\Pi_V(S),$$

où V est un ensemble de voix, S est une séquence temporelle. Son équivalent syntaxique est :

$$S->(voice_1, \ldots, voice_n).$$

Sélection : La sélection sur une séquence temporelle se note algébriquement :

$$\sigma_F(V),$$

où F est une formule, V est un ensemble de voix d'une séquence temporelle. Son équivalent syntaxique est **where. . . contains**.

Exemple : l'expression

$$\Pi_{id,\Pi_{pitch,rythm}(voice)}(\sigma_{\Pi_{lyrics}(voice)\supset'Heureux\ les\ hommes',composer='Faure'}(\textbf{Psalms}))$$

correspond à la requête

from	Psalms
construct	id, voice -> (pitch,rythm)
where	voice -> lyrics **contains** 'Heureux les hommes'
and	composer = 'Faure'

Attention. si la séquence temporelle consiste en plusieurs voix synchronisées, **contains** devra faire figurer autant de conditions que de voix.

Exemple :

from	Psalms
construct	id
where	voice−>(lyrics,pitch) **contains** ('Heureux les hommes', 'A3,B3,B3'))

Produit : Le produit de deux séquences temporelles se note algébriquement :

$$s \times t,$$

où s et t sont deux séquences temporelles. Son équivalent syntaxique est **synch**.

Exemple : l'expression

$$\Pi_{M.Voice \times F.Voice}(\sigma_{M.Id=F.Id}(Male \times Female))$$

correspond à la requête

from	Male M, Female F
let	$duet := **synch**(M.Voice,F.Voice)
construct	$duet
where	M.id=F.id

4.2 Algèbre temporelle

Addition : L'addition sert à propager au niveau séquence une fonction utilisateur (unaire ou binaire) définie au niveau atomique. Elle se note algébriquement :

$$\oplus_{op},$$

où *op* est une fonction utilisateur. Les équivalents syntaxiques sont **map** et **map2**.

On passe en paramètres à **map** la fonction utilisateur *op* et une séquence temporelle. On passe en paramètres à **map2** une fonction binaire *op* et deux séquences temporelles.

Exemples : l'expression

$$\Pi_{\Pi_{\text{pitch}}(\text{voice}) \oplus_{transpose}(1)}(\textbf{Psalms})$$

correspond à la requête

from	Psalms
let	$transpose := **map**(*transpose*(1), voice− >pitch)
construct	$transpose

L'expression

$$\Pi_{\Pi_{\text{trumpet}}(\text{voice}) \oplus_{harm} \Pi_{\text{clarinet}}(\text{voice})}(\textbf{Duets})$$

correspond à la requête

from	Duets
let	$harmonic_progression := **map2**(*harm*, voice− >trumpet, voice− >clarinet)
construct	$harmonic_progression

Composition : La composition sert à modifier le domaine temporel de définition d'une séquence en composant la séquence avec une fonction temporelle (*i.e.* une fonction de $\mathbb{N}$ dans $\mathbb{N}$). Elle se note algébriquement

$$S \circ \gamma,$$

où γ est une fonction temporelle et S une séquence temporelle. L'équivalent syntaxique est **comp**(S, γ).

Exemple l'expression

$$\Pi_{\text{Elec\_Daily}\circ\text{Warp(7)}}(\textbf{Elec\_Daily})$$

correspond à la requête

from	Elec_Daily
let	$Elec_sunday := **comp**(Elec_Daily, Warp(7))
construct	$Elec_sunday

Dérivation - agrégation Cet opérateur procède en deux temps ; dans un premier temps la dérivation génère un ensemble de vues locales de la séquence temporelle, dans un second temps on applique sur chaque vue locale une fonction d'agrégation. La première étape fait appel à une famille de fonctions temporelles, le plus souvent la famille **Shift**.

Ce double opérateur se note algébriquement

$$\mathbf{A}_{\lambda,\Gamma}(\mathrm{S}),$$

où S est une séquence temporelle, Γ est une famille de fonctions temporelles et λ est une fonction d'agrégation.

La famille de fonctions temporelles Γ est une application du domaine temporel dans l'ensemble des fonctions temporelles. Précisément, à chaque instant n, $\Gamma(n) = \gamma$, une fonction temporelle. Les deux familles de fonctions temporelles les plus utilisées sont **Shift** et **Warp**. Elles sont proposées nativement.

Son équivalent syntaxique est

$$\textbf{derive}(S, \Gamma, \lambda),$$

Exemple L'expression

$$\Pi_{id,\mathbf{A}_{\text{dtw(P)},\text{Shift}}}(\textbf{Psalm})$$

correspond à la requête

from	Psalm
let	$dtwVal := **derive**(voice, Shift, dtw(P))
construct	id, $dtwVal

5 Implantation

Pour valider l'utilisabilité du langage, une implantation a été réalisée en OCaml. Par implantation on entend l'écriture d'un interprète lisant en entrée des requêtes et des définitions de fonctions, effectuant le travail demandé et retournant à l'utilisateur le résultat (éventuellement vide) ou un message d'erreur en cas de problème.

5.1 Fonctionnement général

Dans un premier temps, on extrait la plus grosse partie de la requête pouvant être évaluée de manière classique dans les clauses **let**, **construct** et **where**. La requête simplifiée est envoyée au gestionnaire de bases de données qui renvoie un sur-ensemble de la réponse finale. Chaque ligne de l'ensemble intermédiaire est appelée successivement pour être filtrée et traitée le cas échéant.

L'interprète se décompose en quatre parties.

Front-end : Lecture et analyse lexicale et syntaxique de la requête. Création d'un arbre de syntaxe représentant la requête.

Génération de code SQL : Extraction de la partie SQL standard de la requête.

Evalutations non-standards : Gestion des extensions apportées par le langage (non-exprimables en SQL) : manipulation de séquences temporelles, fonctions utilisateurs.

Sortie : Formatage de la réponse et affichage.

5.2 Architecture

Le code est découpé en différents modules.

Ast, Elt : Structure de l'arbre représentant une requête.

Lexer, Parser : Analyse lexicale et syntaxique. Le système vérifie que la syntaxe est correcte et génère l'arbre de syntaxe.

IMySql : Interfaçage avec MySql. On décompose l'arbre représentant la requête totale en deux arbres, l'un représentant la requête SQL et l'autre représentant les calculs à effectuer sur les résultats renvoyés par MySql. La requête MySql est générée au format texte et exécutée.

Printer : Affichage du résultat.

Functions, Eval : Stockage et évaluation des fonctions utilisateurs et pré-définies.

Funlib : Librairie standard de fonctions pré-définies appelables par un utilisateur (average, transpose, min, max, harmonic).

5.3 Structure des tables

Les données sont stockées dans des tables MySQL. N'importe quel SGBD peut être utilisé pourvu qu'une librairie permettant d'interfacer OCaml et ce SGBD existe.

Les séquences temporelles y sont stockées avec le type blob sous forme d'une suite de caractères. Le caractère ; est utilisé comme séparateur entre deux instants de la

séquence. Par convention on appelera *instant* tout élément compris entre deux ;. Le symbole - est le séparateur entre les voix à chaque instant. Autrement dit, deux événements séparés par le symbole - ont lieu au même instant, dans deux voix différentes.

Une séquence temporelle composée de n voix synchronisées débute par un entête décrivant le format du contenu de la séquence temporelle : l'indicateur (TS :n), où n est le nombre de voix synchronisées de la séquence, suivi de n couples (nom - type) décrivant chaque voix de la séquence.

La table 3.1 donne un exemple de structure d'une table pour la collection *Psalms*. La séquence temporelle prend ses valeurs dans le domaine **vocals**, c'est-à-dire **lyrics**×**pitch**.

id_doc (int)	TS1 (blob)
1.1	TS :2 ; lyrics-string ; pitch-int ; Sei-23 ; gneur-23 ; qui-25 ; jus-26 ; qu'i-27 ; cy-20 ; m'a-22 ; es-21 ; té-22 ; ...
2.1	TS :2 ; lyrics-string ; pitch-int ; Heu-24 ; reux-22 ; qui-23 ; n'ou-20 ; vre-22 ; point-18 ; son-20 ; coeur-26 ; ...

TAB. 3.1 – Exemple de table MySQL

5.4 Interface d'interrogation

La figure 3.1 montre une interface d'interrogation, un exemple de requête saisie par un utilisateur et la réponse.

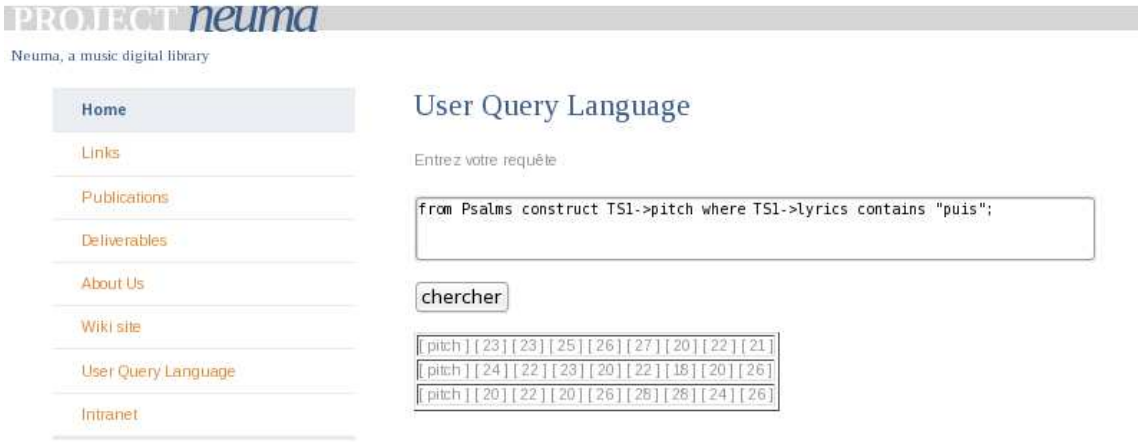


FIG. 3.1 – Interface d'interrogation

Chapitre 4

Un index pour la recherche par contenu dans une collection de sources musicales symboliques

Les travaux exposés dans ce chapitre ont fait l'objet de deux publications : *Indexing Symbolic Music Score* [23] et *The Melodic Signature Index For Fast Content-Based Retrieval of Symbolic Scores* [24].

Dans ce chapitre nous proposons l'utilisation d'un index pour la recherche par contenu dans une grande collection de sources musicales symboliques. Il permet de conduire différents types de recherche sans avoir à changer de structure d'index. Sa particularité repose sur l'encodage de diverses informations qui interviennent dans les différentes recherches. Il s'appuie de manière fondamentale sur la notion de signature algébrique. L'index utilisé est une adaptation de l'AS-index [31] aux données musicales symboliques. Pour la recherche approchée, la similarité est mesurée grâce à la distance n-gram introduite dans [81].

1 Remarques préliminaires

On se place dans le contexte d'une bibliothèque de partitions numériques (DSL), c'est-à-dire une collection de contenu musical symbolique. Une partition peut être interrogée de plusieurs façons : recherche exacte, transposée, avec ou sans rythme, et recherche approchée par motif. On pourrait également envisager une recherche uniquement rythmique ou par rythme relatif simplement en adaptant les principes exposés dans ce chapitre. La recherche par similarité (déterminer des ensembles de partitions similaires) n'est en revanche pas abordée ici.

Définition 31 (Représentation exacte, représentation transposée)**.** *On distingue deux types de réprésentations. La représentation exacte est la suite* absolue *des notes de la partition. La représentation transposée est la suite* relative *des notes, c'est-à-dire les intervalles successifs de la partition. L'information rythmique peut être ou non associée à chacune des représentations.*

Exemple 7. *Soit la représentation exacte (ou* absolue*) :*

<21 ; 22 ; 21 ; 26 ; 26 ; 28 ; 24 >.

La représentation transposée (ou relative*) correspondante est la suivante :*

`<+1 ; -1 ; +5 ; 0 ; +2 ; -4>`.

On remarque que la représentation exacte est équivalente à la représentation transposée avec une information supplémentaire (valeur de la note initiale).

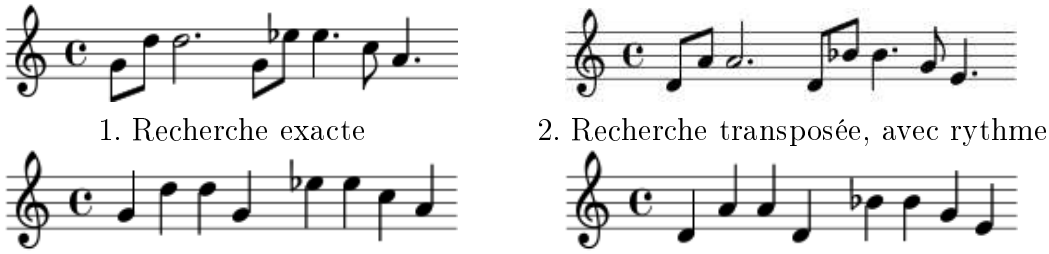
Une approche naïve consiste à stocker autant de représentations que de types d'interrogations possibles, et de construire un index pour chacune. Il est évident que cette solution n'est plus envisageable dès que la collection grandit. Le volume de données à stocker et à indexer est en effet rédhibitoire. De plus, la répétition des informations d'une représentation à l'autre montre que cette approche est contre-productive. Réussir à construire un index unique pour tous les types d'interrogations représente un avantage considérable.

2 Recherche par contenu

Un service de recherche par contenu conçu pour notre collection de ressources musicales symboliques doit prendre en entrée un motif P et renvoyer une partition ayant au moins une voix v, telle qu'elle même ait au moins une position p telle que P *corresponde* au fragment $f_p = v[p]v[p+1]\ldots$. La notion de correspondance est totalement subjective.

Principe de recherche exacte

La figure 4.1 montre quatre motifs qui peuvent répondre au même motif passé en requête, en tant que recherche exacte, recherche transposée avec rythme, recherche mélodique, recherche transposée sans rythme.



1. Recherche exacte
2. Recherche transposée, avec rythme
3. Recherche exacte, sans rythme
4. Recherche transposée, sans rythme

FIG. 4.1 – Quatre types de recherche.

L'interprétation 1 correspond à une recherche exacte avec rythme. Le motif passé en requête doit être représenté précisément dans la partition.

L'interprétation 2 correspond à une recherche transposée avec rythme. Le motif passé en requête et le fragment restitué en réponse ont la même succession d'intervalles, mais pas la même succession de notes. On remarque que l'interprétation 1 répond également à une recherche transposée avec rythme.

L'interprétation 3 est une recherche exacte sans rythme. La succession des notes est la même que celle du motif passé en requête, mais le rythme est ignoré. L'interprétation 1 répond à une recherche exacte sans rythme, mais pas l'interprétation 2.

Enfin, l'interprétation 4 est une recherche transposée sans rythme. On cherche les fragments de voix ayant la même succession d'intervalles que celle du motif recherché, en ignorant le rythme. Cette requête est la plus générale, c'est celle qui aura le plus grand nombre de réponses.

Principe de recherche approchée

La recherche approchée pour une distance d se définit comme suit : pour un motif P donné, et un fragment f_p d'une voix v à une position p, on dit qu'il y a correspondance si $d(P, f_p) \leq \eta$ pour une tolérance d'erreur η définie par l'utilisateur.

La distance que l'on souhaite utiliser pour mesurer la similarité entre deux fragments de voix est la distance d'édition [53], dont on a rappelé la définition dans le chapitre 1. Il est connu que la distance d'édition est difficile à indexer [58]. Notre approche se passe en deux étapes, en suivant une méthode proposée par [19]. Premièrement on fait une recherche approchée pour une distance qui est une *borne inférieure* de la distance d'édition, et pour laquelle on peut utiliser un index. Précisément, nous utiliserons la distance n-gram A_n introduite dans [81] (dont on rappelle la définition dans ce chapitre, section 7 définition 40). Une recherche basée sur cette distance retrouve tous les fragments de voix tels que $A_n(P, f_p) \leq \tau$. Tous les fragments ne vérifiant pas cette inégalité (et donc écartés) ne vérifient *a fortiori* pas $d_{edit}(P, f_p) \leq \tau$.

Cependant, comme A_n est une borne inférieure de la distance d'édition, il peut se produire que $A_n(P, f_p) \leq \tau < d_{edit}(P, f_p)$, autrement dit que f_p soit un faux positif. La deuxième étape accède aux candidats restants et fait une nouvelle recherche, cette fois ci avec la distance d'édition, afin d'écarter les faux positifs.

La distance A_n sert donc de filtre pour la distance d'édition.

3 Principes généraux du MS-index

Le principe de l'index introduit dans [31], appelé AS-index, peut être résumé comme suit : (i) une structure traditionnelle de table de hachage ; (ii) les clés utilisées sont des fragments musicaux de taille fixe, les n-gram, présents dans au moins un document de la collection ; (iii) la fonction de hachage s'appuie sur un calcul de *signature algébrique* des n-grams.

Si les deux premiers aspects sont relativement classiques dans le domaine de l'indexation textuelle [56], le dernier point s'inspire de travaux récents sur le traitement de texte basés sur les signatures algébriques [54, 31]. Dans le présent travail, on les adapte aux spécificités de la musique symbolique.

L'efficacité de cet index repose en grande partie sur les propriétés calculatoires des signatures algébriques. On adapte l'AS-index aux données musicales symboliques pour représenter dans un enregistrement toute l'information musicale nécessaire aux différents types de recherche. On ne parle alors plus de signature algébrique mais signature musicale. Ce nouvel index est appelé MS-index (*Musical Signature Index*).

Remarque. *Il est évident que l'indexation de contenu musical ne peut pas s'envisager de la même manière que l'indexation d'un texte. Une différence inévitable est l'absence de notion de mot dans le contexte musical. Le découpage de la partition en segments de longueurs constantes n permet de réintroduire artificiellement cette notion. En cela on se rapproche beaucoup des techniques utilisées en bio-informatique et recherche génomique [14].*

4 Notations, rappels et définitions

Une partition symbolique est découpée en voix monophoniques, chaque voix est une séquence d'éléments de $\mathcal{E} \times \mathcal{D}$.

Définition 32 (Descripteur)**.** *On appelle descripteur la représentation d'une voix. Un descripteur est encodé textuellement sous la forme* $D =$ `<`e_1`-`d_1 `;` e_2`-`d_2 `; ... ;` e_n`-`d_n`>` *où chaque* $e_i \in \mathcal{E}$ *code un événement et chaque* $d_i \in \mathcal{D}$ *sa durée.*

Dans ce qui suit $\epsilon(D)$ désigne une suite d'événements, $\tau(D)$ la suite des intervalles dans $\epsilon(D)$, et $\rho(D)$ la suite des durées de D.

Autrement dit, $\epsilon(D)$ représente le profil mélodique de la voix, $\tau(D)$ l'évolution relative des notes et $\rho(D)$ le profil rythmique de la voix. Toutes les combinaisons de ces représentations sont possibles.

Un événement musical est représenté par un élément d'un *corps fini*.

On rappelle qu'un corps fini est un ensemble fini muni de deux lois internes notées $+$ et $\times$, associatives, commutatives et distributives, telles que tous les éléments de l'ensemble ont un inverse sauf l'élément neutre additif. On note $\mathbb{K}$ un corps fini.

Définition 33 (Elément primitif)**.** *Soit* $\mathbb{K}$ *un corps fini. Un élément primitif de* $\mathbb{K}$ *est un élément* α *tel que les puissances successives de* α *énumèrent tous les éléments de* $\mathbb{K}$*.*

Un élément primitif n'est pas nécessairement unique.

On choisit le corps finis à 256 éléments pour l'encodage des événements musicaux. Ceci permet l'encodage de 12 octaves de 12 notes, ainsi que celui des silences, fins de phrases, et autres caractères particuliers.

On introduit à présent une suite de définitions de *signatures algébriques* ainsi que leurs propriétés. C'est sur ces propriétés fondamentales que repose l'efficacité de notre index. Dans ce qui suit, on désigne par $D = e_0 e_1 \dots e_{M-1}$ un descripteur encodant une séquence de M événements, considérés comme des éléments de $\mathbb{K}$.

Définition 34 (AS-signature)**.** *Soit* $\mathbb{K}$ *un corps fini et* α *un élément primitif. La* α*-signature algébrique d'un descripteur* D *est définie par*

$$AS_\alpha(D) = e_0 + e_1.\alpha + e_2.\alpha^2 + \ldots + e_{M-1}.\alpha^{M-1}$$

Définition 35 (AS_m-signature)**.** *Soit* $\mathbb{K}$ *un corps fini et* $\alpha_1, \dots \alpha_m$ *m éléments primitifs distincts. Alors la m-signature algébrique d'un descripteur* D *est obtenue en concaténant l'ensemble des* α_i*-signatures. On obtient ainsi une signature de taille* m*.*

On s'intéresse à la signature partielle d'un descripteur. Les propriétés calculatoires de cette signature sont un des piliers de l'index.

Définition 36 (Signature cumulée CAS). *Soit $l \in [0, M-1]$ une position de D. La signature algébrique cumulée $CAS(D, l)$ en l est la signature algébrique du préfixe de D finissant en e_l, c'est-à-dire*

$$CAS(D, l) = AS(e_0 \dots e_l).$$

Enfin, on s'intéresse au cas particulier des n-grams tirés d'un descripteur D.

Définition 37 (Signature partielle et signature n-gram). *La signature algébrique partielle de l à l' est la signature de la sous-chaîne extraite de D entre les positions l et l' :*

$$PAS(D, l, l') = AS(e_l e_{l+1} \dots e_{l'}).$$

En particulier, on s'intéresse aux sous-chaînes de longueur n :

$$NAS(D, l) = PAS(D, l-n+1, l).$$

Lorsqu'aucune confusion n'est possible, on ne précise pas le descripteur D.

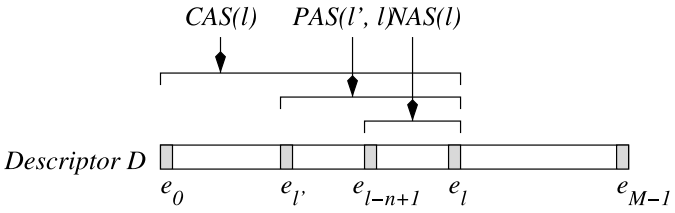


FIG. 4.2 – Différentes signatures d'un descripteur.

La figure 4.2 récapitule les liens entre les différentes signatures.

Les propriétés suivantes permettent de calculer de manière incrémentale les différentes signatures lors de l'indexation d'une partition ou du pré-traitement d'un motif de recherche. La dernière propriété est utilisée de manière cruciale lors de la recherche effective.

Proposition 1. *Soient $\mathbb{K}$ un corps et α un élément primitif. On a les égalités suivantes :*

1. $CAS(l) = CAS(l-1) + e_l \alpha^l$
2. $NAS(l) = \frac{NAS(l-1) - e_{l-n}}{\alpha} + e_l . \alpha^{n-1}$
3. $CAS(l) = CAS(l') + \alpha^{l'+1} PAS(l'+1, l)$

Ces propriétés se vérifient très simplement et ne nécessitent pas de démonstration.

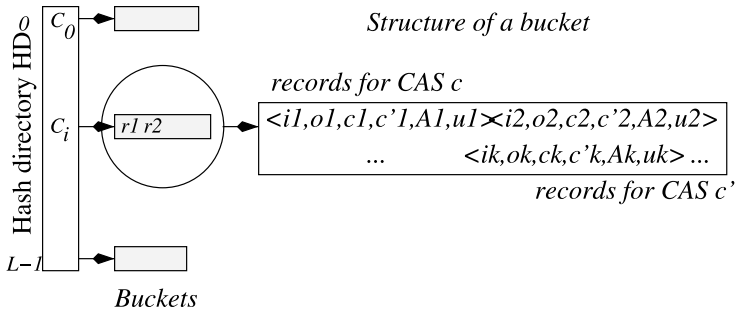


FIG. 4.3 – Structure de l'index

5 Création du MS-index

L'index de signature musicale (MS-index) a une structure classique de table de hachage, qu'on note $HD[0 \dots L-1]$ de taille $L = 2^c$. Les éléments de HD renvoient vers des lignes de taille variable. Chaque ligne contient une liste d'enregistrements qui indexe chacun un n-gram présent dans au moins un descripteur.

On détaille à présent la construction de l'index.

On commence par répertorier de manière exhaustive les n-grams contenus dans la collection (c'est-à-dire qu'on fait l'inventaire de chaque n-gram de chaque voix de chaque partition de la collection). Sur chaque n-gram ainsi exhibé, on calcule plusieurs signatures qui vont déterminer l'organisation de l'index.

Soit G un n-gram tiré de $\epsilon(D)$, et τ la fonction transformant un n-gram de notes en un $(n-1)$-gram d'intervalles de notes (ou progression relative des notes). Soit $S = h_L(G) = AS_m(\tau(G))$. On calcule alors i l'index de la ligne qui réfère à G par

$$i = S \bmod L.$$

En $HD[i]$, on insère l'enregistrement suivant.

Définition 38. *Soit G un n-gram à la position p d'un descripteur D. L'enregistrement indexant G est un 6-uplet $(id(D), p, c_\epsilon, c_\rho, AS_\rho, \perp)$ où*

1. $c_\epsilon = CAS(\epsilon(D), p)$
2. $c_\rho = CAS = (\rho(D), p)$
3. $AS_\rho = AS_m(\rho(D), p)$
4. $\perp$ *est la note la plus basse de G*

Plusieurs choses sont à remarquer. Premièrement on ne calcule qu'une seule signature sur le n-gram G, qui porte sur sa représentation transposée (ou relative). La fonction de hachage h_L est la fonction qui à un n-gram G associe la signature algébrique de sa représentation transposée. Ensuite, les signatures stockées dans l'enregistrement correspondant sont les signatures *cumulées*, c'est-à-dire les signatures du descripteur tronqué à la position p.

On va voir que ces informations sont suffisantes pour réaliser tous les types de requêtes définis plus haut, transposée, exacte, avec ou sans rythme.

6 Recherche exacte de partitions

6.1 Déroulement d'une recherche exacte

On cherche les occurrences d'un motif $P = e_0 e_1 \dots e_{M-1}$ passé en requête dans une collection de partitions symboliques.

On commence par effectuer un pré-traitement sur P. On appelle S_1 le premier n-gram de P, S_2 le dernier n-gram de P et S_p le suffixe de P à partir de S_1.

On calcule les signatures respectives $i_1 = h_L(S_1) = AS_m(\tau(S_1))$, $i_2 = h_L(S_2) = AS_m(\tau(S_2))$ et $AS(S_p, n, M-1)$.

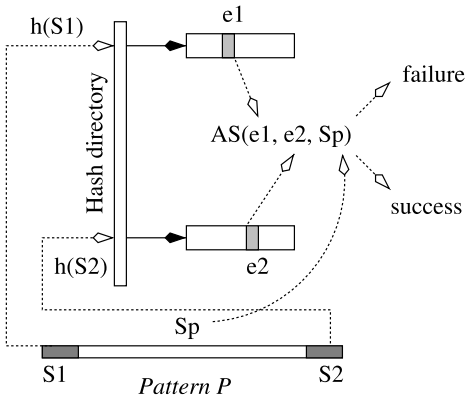


FIG. 4.4 – Recherche exacte utilisant le MS-index.

On fait deux appels à l'index pour récupérer les enregistrements correspondants (en signature) à S_1 et S_2. La ligne $D[i_1]$ contient les entrées

$$(id(D), p_1, c_{1\epsilon}, c_{1\rho}, AS_{1\rho}, \perp_1),$$

de même la ligne $D[i_2]$ contient les entrées

$$(id(D), p_2, c_{2\epsilon}, c_{2\rho}, AS_{2\rho}, \perp_2).$$

Dans les listes récupérées, on sélectionne les enregistrements qui sont dans les mêmes voix et à bonne distance, grâce à une comparaison des identifiants et des positions. On a ainsi les descripteurs D présentant un fragment dont les n-grams initiaux et finaux correspondent en signature aux n-grams initiaux et finaux du motif P recherché. Si l'on veut effectuer une recherche exacte non-transposée, on vérifie l'égalité entre la valeur minimale de S_1 et $\perp_1$.

On vérifie ensuite que

$$c_{2\epsilon} = c_{1\epsilon} + \alpha^{o_1+1} PAS(\epsilon(S_p), p_1 + 1, p_2)$$

Ce calcul permet de vérifier la correspondance, au moins en signature, entre S_p et le suffixe de l'extrait de D.

Pour une recherche avec rythme, on doit ajouter les vérifications supplémentaires $AS(\rho(S_1)) = AS_{1\rho}$, $AS(\rho(S_2) = AS_{2\rho}$ et

$$c_{2\rho} = c_{1\rho} + \alpha^{p_1+1} PAS(\rho(S_p), p_1 + 1, p_2).$$

Ceci termine la recherche exacte.

6.2 Algorithmes

On présente ici le pseudo-code d'une recherche transposée sans rythme.

Input : un motif $P = p_0 \dots p_{K-1}$, la taille n d'un n-gram
Output : la liste des descripteurs contenant $\tau(P)$
// Phase de pré-traitement;
$S_1 := NAS_m(P, n-1)$;
$S_2 := NAS_m(P, K-1)$;
$S_p := PAS_1(P, n, K-1)$;
// Première ligne de l'index;
$i := h_L(S_1)$;
// Seconde ligne de l'index;
$i' := h_L(S_2)$;
// Phase de traitement;
foreach $(id(D), p_1, c_{1\epsilon}, c_{1\rho}, AS_{1\rho}, \perp_1)$ *de* $D[i]$ **do**
  $c_{2\epsilon} = c_{1\epsilon} + \alpha^{p_1+1} \cdot S_p$
  $p_2 = (p_1 + K - n)$ *mod* $(2^c - 1)$;
  if *il existe* $(id(D), p_2, c_{2\epsilon}, c_{2\rho}, AS_{2\rho}, \perp_2)$ *dans* $D[i']$ **then**
    D est candidat
  end
end

Algorithme 4: Recherche transposée, sans rythme

Pour une recherche exacte, on ajoute un test dans la phase de traitement.

Input : un motif $P = p_0 \dots p_{K-1}$, la taille n d'un n-gram
Output : la liste des descripteurs contenant $\tau(P)$
// Phase de pré-traitement;
$S_1 := NAS_m(P, n-1)$;
$S_2 := NAS_m(P, K-1)$;
$S_p := PAS_1(P, n, K-1)$;
// Première ligne de l'index;
$i := h_L(S_1)$;
// Seconde ligne de l'index;
$i' := h_L(S_2)$;
// plus basse note de S_1;
$min_1 := min(S_1)$;
// Phase de traitement;
foreach $(id(D), p_1, c_{1\epsilon}, c_{1\rho}, AS_{1\rho}, \perp_1)$ *de* $D[i]$ **do**
  if $min_1 = \perp_1$ **then**
    $c_{2\epsilon} = c_{1\epsilon} + \alpha^{p_1+1} \cdot S_p$
    $p_2 = (p_1 + K - n)$ *mod* $(2^c - 1)$;
    if *il existe* $(id(D), p_2, c_{2\epsilon}, c_{2\rho}, AS_{2\rho}, \perp_2)$ *dans* $D[i']$ **then**
      D est candidat
    end
  end
end

Algorithme 5: Recherche exacte, sans rythme

6.3 Remarques

Quelle que soit la longueur du motif passé en requête, le nombre d'appels à l'index reste constant, au nombre de deux. Ceci est un avantage en particulier pour la recherche de motifs longs, qui peut se faire en temps constant.

Cet index est de nature probabiliste, en effet les correspondances sont des correspondances en signature et les collisions existent. Cela étant, le nombre de faux-positifs est négligeable, et tend vers zéro lorsque n augmente.

7 Recherche approchée de partitions

7.1 Définitions

Nous donnons pour commencer une série de définitions exposées dans [81]. Nous gardons volontairement des notations générales et non spécifiques au contexte des ressources musicales symboliques.

Soient Σ un alphabet fini, $\Sigma^\star$ l'ensemble de toutes les chaînes d'éléments de Σ et Σ^n l'ensemble des chaînes de longueur n (n-grams) de Σ.

Définition 39 (Profil n-gram)**.** *Soit $P = a_1 a_2 \dots a_N$ une chaîne de $\Sigma^\star$ et soit v dans Σ^n un n-gram. Si $a_i a_{i+1} \dots a_{i+n-1} = v$ pour un certain i, alors P a une occurrence de v. Soit $G(P)[v]$ le nombre total d'occurences de v dans P. Le profil n-gram de x est le vecteur $G_n[P] = (G(P)[v])_n, v \in \Sigma^n$.*

Pour deux chaînes P et Q, la distance n-gram $A_n(P, Q)$ [81] est basée sur le nombre de n-grams communs entre P et Q. Plus les n-grams communs sont nombreux, meilleure est la distance. L'ordre dans lequel apparaissent les n-grams n'est pas important dans cette définition. En revanche le nombre d'occurences des n-gram intervient.

Définition 40. *Soient P, Q des chaînes de $\Sigma^\star$ et soit $n > 0$ un entier. La distance n-gram entre P et Q est*

$$A_n(P, Q) = \Sigma_{v \in \Sigma^n} |G(P)[v] - G(Q)[v]|.$$

A_n n'est pas strictement une distance puisque $A_n(P, Q)$ peut valoir 0 même lorsque $P \neq Q$. Toutefois, le résultat suivant montre que ce cas est peu fréquent dans la pratique.

Proposition 2 (Ukkonen)**.** *Soit P une chaîne de $\Sigma^\star$ et soit $n > 0$ un entier. Si P contient au plus une occurence de chaque $n-1$-gram, alors l'ensemble des chaînes Q telles que $A_n(P, Q) = 0$ est réduite à P.*

Remarquons que cette distance permet de comparer des chaînes de tailles différentes.

7.2 Liens avec la distance d'édition

On peut minorer la distance d'édition par la distance A_n de la manière suivante.

Théorème 3 (Ukkonen). *Pour toute chaîne P, Q,*

$$A_n(P,Q)/(2n) \leq d_{edit}(P,Q).$$

La distance $A_n(P,Q)$ peut être évaluée en temps $O(|P| + |Q|)$, alors que la distance d'édition a pour complexité $O(|P| \star |Q|)$, où $|P|$ et $|Q|$ désignent les longueurs de P et Q respectivement.

On donne à présent un exemple utilisant la distance A_2.

Exemple 8. *Soit D un descripteur et*

$$\epsilon(D) =< 34, 38, 38, 34, 39, 39, 34, 40 \ldots > .$$

On fait une recherche par similarité pour le motif $P =< 34, 38, 39, 39 >$.

P a trois 2-grams, $P_1 =< 34, 38 >$, $P_2 =< 38, 39 >$ et $P_3 =< 39, 39 >$. Son profil 2-gram est $G_2[P_1] = 1$, $G_2[P_2] = 1$, $G_2[P_3] = 1$.

La meilleure approximation de P est $\epsilon(D_{[1-6]}) =< 34, 38, 38, 34, 39, 39 >$ a une distance $A_2(P, \epsilon(D_{[1-6]}) = 2$.

Au contraire, $\epsilon(D_{[4-6]}) =< 34, 39, 39 >$ est une approximation moins bonne avec $A_2 = 3$.

Si l'on compare avec les distances obtenus en utilisant la distance d'édition, on trouve pour la première approximation a $d_{edit}(P, \epsilon(D_{[1-6]}) = 2$ (deux suppressions de notes) et seulement 1 pour la seconde (une insertion de note).

A présent on définit la notion de recherche approchée basée sur la distance A_n.

Définition 41. *Soit $T = t_1 \ldots t_N$ un texte, $P = p_1 \ldots p_m$ un motif, et n un entier $0 \leq n \leq m$. Soit d_i la plus petite distance entre P et la sous-chaîne de T qui commence en t_i, c'est-à-dire $d_i = \min_{i+1 \leq j \leq N} A_n(P, t_i \ldots t_j)$. De plus, soit e_i la position de fin de la plus longue sous-chaîne de T qui réalise d_i. Le problème de recherche approchée pour la distance A_n est de trouver tous les couples (d_i, e_i) pour $1 \leq i \leq N$.*

7.3 Déroulement d'une recherche approchée

L'évaluation d'une recherche approchée se déroule de la manière suivante. Pour alléger les notations, on considère que la collection ne contient qu'un seul descripteur, la généralisation étant immédiate.

On commence par pré-traiter le motif P pour obtenir tous les n-grams P_1, P_2, $\ldots P_q$ présents dans P. Le nombre d'occurences d'un n-gram P_i est noté $G_P[P_i]$ comme dans la définition 39, ou, lorsqu'aucune confusion n'est possible, simplement $G[P_i]$. On accède alors à l'index qui récupère la liste des P_i présents dans le descripteur et leurs positions respectives.

Sans nuire à la généralité, on suppose que tous les n-grams de P qui apparaissent dans le descripteur sont les M premiers n-grams de P, P_1, P_2, $\ldots P_M$. De même, on suppose de plus que le premier n-gram apparaissant dans le descripteur est P_1 et le dernier P_M.

On fusionne toutes les listes de positions en une seule liste ordonnée et on obtient

$$list = \{p_1^1, p_1^2 \ldots p_j^i \ldots p_{last}^M\}$$

où p_j^i est la j-ème position du n-gram P_i dans le descripteur. Notons que, pour un j donné, il est impossible de connaitre la manière dont sont ordonnés p_j^i et p_j^k, étant donné que les n-grams peuvent apparaitre n'importe où dans le descripteur et pas nécessairement dans le même ordre que dans le motif.

Ensuite, on applique une fenêtre glissante de longueur L sur la liste. La proposition suivante nous permet de borner la taille de L.

Proposition 4. *Soit $P = a_1 \dots a_m$ un motif, et p une position dans un descripteur D. La distance A_n minimale entre P et l'extrait de D débutant en p se trouve dans une fenêtre de taille au plus $L = 2m - n + 1$.*

Preuve :

Un motif P de longueur m a $m-n+1$ n-grams. Une fenêtre de taille $L = 2m-n+1$ a $2m - 2n + 2$ n-grams, parmi lesquels au plus $m - n + 1$ n'appartiennent pas à P. Pour des fenêtres de taille plus grande que L, les n-grams n'appartenant pas à P seront toujours en nombre supérieur à ceux qui lui appartiennent. □

Pour chaque position dans la fenêtre, on détermine (d_i, e_i) où i est une position dans la liste (en d'autre termes, il existe j, k tels que $i = p_j^k$) et e_i (position finale) est une position de la fenêtre. Nous illustrons cet algorithme par un exemple.

Exemple 9. *Soit D un descripteur et*

$$\epsilon(D) =< 34, 39, 39, 34, 34, 40, 34, 40 \dots > .$$

On fait une recherche par similarité pour le motif $P =< 34, 39, 34, 40 >$.

Le motif P a trois 2-grams, $P_1 =< 34, 39 >$, $P_2 =< 34, 40 >$ et $P_3 =< 39, 34 >$, et le profile 2-gram de P est $G[P_1] = 1$, $G[P_2] = 1$, $G[P_3] = 1$.

Après accès à l'index, la liste fusionnée est

$$\mathcal{L} =< p_1^1, -, p_1^3, -, p_1^2, -, p_2^2, -, \dots >$$

On applique une fenêtre glissante de taille $L = 8-4+1 = 5$ sur $\mathcal{L}$. Pour $[p_1^1, -, p_1^3, -, p_1^2]$, la distance minimum est $A_2 = 2$, et la plus longue sous-chaîne qui réalise cette distance se termine au 2-gram qui commence à la position p_1^2. Précisément, la plus longue sous-chaîne réalisant la distance minimum est $< 34, 39, 39, 34, 34, 40 >$ (avec trois 2-grams communs et deux 2-grams n'appartenant pas à P). On renvoie le couple $(d_1, e_1) = (2, 5)$.

On glisse la fenêtre à la position suivante de la liste. Pour la fenêtre $[p_1^3, -, p_1^2, -, p_2^2 -]$ la distance minimum est $A_2 = 2$, et la plus longue sous-chaîne réalisant cette distance est $< 39, 34, 34, 40 >$. On renvoie le couple $(d_3, e_3) = (2, 5)$, on glisse jusqu'à la position suivante et ainsi de suite.

7.4 Algorithme

Le pseudo-code qui suit résume l'algorithme de recherche approchée.

A chaque déplacement de la fenêtre, la distance est initialisée à $m - n$, qui est la distance entre P et le premier n-gram rencontré appartenant à P (en effet, P possède $m - n + 1$ n-grams, et on positionne la fenêtre sur un n-gram de P présent dans D). La distance augmente ou diminue en fonction des n-grams rencontrés dans la fenêtre.

On comptabilise le nombre de n-grams rencontrés jusqu'à un certain point dans la fenêtre en remplissant un vecteur C tel que $C[h] = G[P_h]$.

On comptabilise également les blancs dans une variable n_b.

On garde un pointeur **next** sur la liste de manière à savoir quel est la position suivante pour la borne gauche de la fenêtre (en d'autre termes s'il existe des blancs entre deux positions de la liste, ils sont ignorés lorsque l'on glisse la fenêtre). Quand un indice j est dans la liste, cela signifie qu'il existe $h \in [1, M]$ tel que $j = p^h$.

Input : un motif $P = p_0 \dots p_{K-1}$, la taille n d'un n-gram
Output : la liste des descripteurs contenant une approximation de P
$i_{left} = p_1^1$;
while $i_{left} < p_{last}^M$ **do**
  $d_{min} = m - n$; $C[1:M] = 0$; $n_b = 0$; $e = i_{left}$; $i_{right} = i_{left} + L$;
  for $j = i_{left} + 1$ *to* i_{right} **do**
    if j *in list* $(j = p^h)$ **then**
      $C[h] = C[h] + 1$
    else
      $n_b = n_b + 1$
    end
    $d = \Sigma_{h=1}^{M} |G(h) - C(h)| + n_b$;
    if $d \leq d_{min}$ **then**
      $e = j$; $d_{min} = d$
    end
  end
  $(d_{i_{left}}, e_{i_{left}}) = (d_{min}, e)$;
  $i_{left} = \text{next}(list)(i_{left})$;
end

Algorithme 6: Recherche approchée

8 Résultats expérimentaux : recherche exacte

8.1 Description des données

On construit une bibliothèque de partitions au format MusicXML à partir de collections accessibles en ligne (voir tableau 4.1).

La taille des descripteurs varie grandement d'une collection à l'autre. La taille moyenne (toutes collections confondues) est de 967 bytes, et varie de 444B (en moyenne) pour la collection `bach`, jusqu'à 7,020B (en moyenne) pour la collection `gutenberg`.

Le rapport *taille du descripteur/taille du document* varie de 9 ‰ dans `wikifonia` à 23 ‰ dans `hymns`.

[1] `bach` : `www.jsbchorales.net`
[2] `gutenberg` : `www.gutenberg.org/wiki/Gutenberg:The\_Sheet\_Music\_Project`
[3] `hausmusik` : `www.hausmusik.ch`
[4] `musicxml` : `www.musicxml.org`
[5] `wikifonia` : `www.wikifonia.org`

collection	# files	files size	# desc.	desc. size
bach[1]	280	27.1 MB	1,243	539 KB
guttenberg[2]	137	197.2 MB	352	2,413 KB
hausmusik[3]	452	140.9 MB	1,218	1,944 KB
hymns	1,752	84.6 MB	3,885	1,954 KB
musicxml[4]	405	38.9 MB	1,738	713 KB
wikifonia[5]	3,583	302.7 MB	3,570	2,787 KB
wima[6]	961	427.3 MB	3,110	4,624 KB
misc[7]	94	8.9 MB	101	89 KB
all	7,664	1,227.6 MB	15,517	15,063 KB

TAB. 4.1 – Collections de partitions MusicXML

8.2 Temps de construction et taille d'un index

collection	temps de construction	taille de l'index
bach	0.7 s	1.0 MB
gutenberg	3.3 s	5.1 MB
wima	11.4 s	9.5 MB
all (3-gram)	206.6 s	28.5 MB
all (4-gram)	82.6 s	29.7 MB
all (5-gram)	47.0 s	31.3 MB
all (6-gram)	35.9 s	33.2 MB
all (7-gram)	33.2 s	35.1 MB

TAB. 4.2 – Temps de construction et taille de l'index par collection

Le tableau 4.2 répertorie le temps de construction et la taille de l'index pour les différents jeux de données. Les différents critères ayant une influence sont la taille des descripteurs et la longueur des n-gram (choix de n). On effectue des comparaisons sur ces deux critères.

On commence par étudier les collections `bach`, `gutenberg` et `wima` avec $n = 4$. La première constatation est que le temps de construction ne croît pas linéairement en fonction de la taille des descripteurs. Par exemple la collection `gutenberg`, dont les descripteurs font en moyenne la moitié de la taille de ceux de la collection `wima`, ne nécessite que le tiers du temps de construction. De même, toujours avec $n = 4$, la collection `all` dont les descripteurs sont de taille trois fois supérieure à ceux de la collection `wima`, prend 7,5 fois plus de temps pour la construction de son index.

Ceci est dû entre autre à la gestion des collisions lors du remplissage de la table de hachage.

En revanche et comme on peut s'y attendre, la taille de l'index grandit linéairement relativement à la taille des descripteurs. Choisir un n plus grand n'a qu'un impact mineur sur la taille de l'index, mais un impact important sur le temps de construction. Par exemple, un index 7-gram nécessite 25% d'espace en plus qu'un index 3-gram, mais 7 fois moins de temps de construction.

[6] `wima` : `www.icking-music-archive.org`

[7] `hymns` : `www.hymnsandcarolsofchristmas.com`

coll.		Tr	Tr+Ry	Ex	Ex+Ry
gutenberg	MS-index	38.1	27.8	36.9	32.4
	Sc	323.1	212.2	293.4	302.1
	speed-up	8.5	7.6	7.9	9.3
wima	MS-index	10.4	7.5	9.7	7.5
	Sc	637.4	432.1	581.1	595.2
	speed-up	61.3	57.6	59.9	79.3
all	MS-index	41.6	20.7	33.3	24.5
	Sc	2,514.2	1,490.2	2,305.3	2,030.1
	speed-up	60.4	72.0	69.2	82.9

Tab. 4.3 – Impact de la taille du jeu de données sur le temps de recherche (ms)

8.3 Temps de recherche en fonction de n

La figure 4.5 montre que plus on choisit un n grand pour le découpage en n-grams, plus la recherche est rapide, quelle que soit l'interprétation choisie (transposée, exacte, avec ou sans rythme). Des n-grams plus grands signifient moins de collisions et donc moins d'enregistrements par ligne.

L'amélioration la plus significative a lieu quand on passe du 3-gram au 4-gram. En effet, les 3-gram donnent un nombre considérable de collisions et donc un très grand nombre d'enregistrements par ligne. Les 4-grams sont suffisamment discriminants pour diviser le temps de recherche par 2.

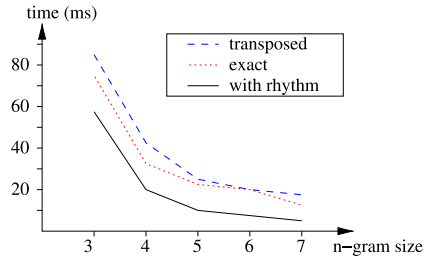


Fig. 4.5 – Impact de la taille du n-gram sur le temps de recherche

Les différences de performance entre les recherches exactes, transposées ou sans rythme sont dues en grande partie à la sélectivité des critères de recherche.

Au contraire de la recherche transposée (Tr) où toutes les comparaisons en signature doivent être effectuées, la recherche exacte (Ex) commence par éliminer une grande partie des candidats par une simple comparaison de la première note du n-gram. En conséquence, le nombre de comparaisons de signatures à effectuer diminue. Si l'on ajoute les critères de rythme, recherches exacte et transposée ont des peformances similaires, et sont plus rapides que les recherches Tr et Ex sans rythme puisqu'on ajoute un filtre sur les enregistrements en ajoutant un calcul de signature supplémentaire.

8.4 Temps de recherche en fonction des descripteurs

Enfin, le tableau 4.3 compare les performances en temps de recherche par rapport à celles d'un scan complet.

Notre index a de meilleures performances que le scan complet sur tous les jeux de données (le rapport variant de 800% à 10 000%). Le temps de recherche avec le MS-index ne dépend pas de la taille des descripteurs : `wima` est deux fois plus grand que `gutenberg` mais les recherches se font en quatre fois moins de temps. On trouve le même rapport quand on compare à `all`, dont la taille est trois fois supérieure.

Les performances de l'index sont en revanche sensibles à la distribution des données. En effet, une répartition asymétrique entraine des lignes ayant un grand nombre d'enregistrements, et en conséquence un plus grand nombre de tests par recherche.

Les recherches avec rythme sont plus rapides puisqu'elles proposent un filtre supplémentaire sur les enregistrements rapatriés, évitant ainsi les comparaisons inutiles.

On constate que le gain de temps est plus faible pour la collection `gutenberg` que pour les autres collections. Une explication est le fait que celle ci présente quelques très grands fichiers, générant donc plus d'enregistrements par documents. Etant donné que l'identifiant du document est aussi un critère de filtre (les comparaisons ne se font que pour des enregistrements ayant les mêmes identifiants), plus de tentatives de mise en correspondance sont menées.

9 Résultats expérimentaux : recherche approchée

Pour valider notre algorithme de recherche approchée, on commence par fixer un seuil τ pour la distance n-gram A_n. On dit qu'une séquence S_i est une réponse (aussi bien pour la recherche se servant du MS-index que pour la recherche exhaustive) pour un motif P si cette séquence vérifie $A_n(P, S_i) \leq \tau$.

Les résultats sont présentés dans le tableau 4.4, pour des motifs de longeur 12 et un seuil τ égale à 5.

collection	MS-Index (ms)	Exhaus. (ms)	Accélération	A_n	d_{edit}	f/p (‰)
`bach`	0.5	115.2	230.4	4.61	4.60	0.11
`guttenberg`	5.2	498.3	95.8	4.05	4.01	0.06
`wima`	6.7	1,005.3	150.0	4.44	4.42	0.14
`all`	16.0	4,558.0	284.9	4.01	3.97	0.27

TAB. 4.4 – Recherche approchée pour les différentes collections

On remarque que la recherche approchée est plus rapide que la recherche exacte, et l'accélération est par exemple de 285 pour la collection `all` rassemblant l'ensemble des partitions.

Rappelons que pour la recherche approchée, on parcourt une liste unique de positions, obtenue en fusionnant toutes les positions rappelées après accès au MS-index, ce qui garantit que chaque position n'est vérifiée qu'une seule fois. En conséquence, plus la liste de positions est courte, plus la recherche est rapide. Ceci s'illustre en particulier lorsque l'on effectue une recherche dans `all`, qui prend trois fois plus de temps que la même recherche dans `wima`.

9.1 Lien entre distance A_n et distance d'édition

Un résultat important mis en évidence par nos tests est que les distances A_n et d_{edit} sont dans la plupart des cas très proches. En effet, à l'exception de quelques motifs, les deux distances sont les mêmes.

Comme on l'a rappelé dans la partie 7.2, la distance A_n peut en théorie être utilisée comme borne inférieure à la distance d'édition. Cependant, dans le cadre particulier de notre bibliothèque numérique de partitions, la similarité des deux distances est plus intéressante.

La distance A_n fournit donc une approximation raisonnable de la distance d'édition, avec l'avantage considérable que la distance A_n peut être indexée et utilisée efficacement pour la recherche approchée.

configuration	MS-Index (ms)	Exhaus. (ms)	Accélération	A_n	d_{edit}	f/p (‰)
4-grams,$\tau = 4$	15.8	4,570.6	289.3	2.77	2.73	0.27
4-grams,$\tau = 5$	16.0	4,558.0	284.9	4.01	3.97	0.27
4-grams,$\tau = 6$	16.6	4,518.2	272.2	5.34	5.31	0.29
4-grams,$\tau = 7$	17.5	4,472.0	255.5	6.59	6.57	0.70
3-grams,$\tau = 5$	57.1	4,644.1	81.3	4.05	3.92	0.26
5-grams,$\tau = 5$	7.2	4,053.1	562.9	4.01	3.99	0.27

TAB. 4.5 – Recherche approchée pour différentes valeurs de n et τ

Les résultats rassemblés dans le tableau 4.5 montrent que la différence entre A_n et d_{edit} varie de 0.5% à 2.5%.

On observe également l'influence des choix de τ et n. On remarque que la valeur de τ n'a pas d'impact sur le temps de recherche, aussi bien pour le MS-index que pour le scan exhaustif.

Tout à fait logiquement, les deux distances suivent en moyenne l'augmentation de la valeur de τ, au sens où plus τ augmente, plus on a de réponses.

En revanche, si l'on augmente la valeur de n, le temps de réponse diminue.

9.2 Faux positifs

Comme tout index basé sur une table de hachage, le MS-index est amené à gérer des collisions pour ses entrées, ce qui entraine l'éventualité de faux-positifs (f/p). En conséquence, une recherche utilisant le MS-index devrait inclure un parcours final sur l'ensemble des réponses retournées si l'on veut impérativement écarter les faux-positifs.

Le taux des faux positifs est un indicateur pertinent de la qualité de l'index. Dans les tableaux 4.4 et 4.5, on peut observer que le taux de faux positifs est particulièrement faible, soit 0.70 ‰dans le pire des cas.

Pour la recherche approchée, les faux positifs apparaissent lorsqu'un n-gram du motif recherché a la même signature qu'un n-gram absent. Les faux positifs peuvent donc apparaitre lorsque l'on calcule la distance A_n en utilisant le MS-index, mais pas lors d'un parcours exhaustif.

Le tableau 4.4 met en évidence un taux plus faible de faux positifs dans la collection `guttenberg`. Ceci s'explique par un nombre de collisions nettement inférieur à celui des autres collections.

10 Conclusion

Dans ce chapitre, nous avons décrit une approche pratique de l'indexation de la recherche par contenu dans une grande collection de partitions.

L'index ainsi conçu possède plusieurs caractéristiques qui en font un bon choix pour l'indexation d'une grande collection de partitions : (i) *flexibilité* : une structure d'index unique permet de conduire différents types de recherche, (ii) *compacité* : malgré la richesse de l'information vers laquelle il renvoie, l'index ne requiert pas beaucoup d'espace relativement au stockage de la collection globale et (iii) *efficacité* : quelques millisecondes suffisent à renvoyer un résultat, même pour des motifs longs cherchés dans de grandes collections.

En conduisant des expériences sur des données réelles, il est apparu que la distance A_n et la distance d'édition sont en réalité très proches. En effet, à l'exception de quelques motifs isolés, les deux distances renvoient les mêmes résultats. En théorie, A_n peut être utilisé comme une borne inférieure très large de la distance d'édition. En réalité, on peut s'en servir comme d'une très bonne approximation, avec l'avantage considérable que A_n peut être indexée.

Chapitre 5

Application : plateforme NEUMA

Le contenu de ce chapitre reprend et approfondi la publication *The* NEUMA *Project : Towards Cooperative On-line Music Score Libraries* [1].

La plateforme NEUMA a pour but de mettre du contenu musical symbolique à disposition de communautés d'experts, notamment des musicologues, historiens de la musique et éditeurs, par le biais d'une plateforme collaborative. L'ambition de cette plateforme est de proposer plus qu'un simple outil de stockage et d'affichage de grandes collections de partitions numériques, en offrant entre autres des outils d'interrogation et d'analyse à l'échelle de plusieurs collections.

1 Introduction

1.1 Contexte et motivation

La plupart des plateformes actuellement en service reposent sur l'utilisation du format audio. Celui ci a l'avantage de la compacité qui rend diffusion et stockage très facile, ainsi qu'un accès immédiat au contenu (écoute). En revanche il est beaucoup plus compliqué – voire impossible – d'en extraire une information structurée.

La représentation symbolique offre au contraire une représentation très détaillée, qui peut s'exploiter de manière tout autre que la même partition au format audio. L'information contenue par une partition sous sa forme symbolique est en effet très riche. Une partie de ces informations concerne l'affichage (ou gravure) de la partition. Le reste concerne le contenu musical au sens strict : valeur et durées des notes, paroles, instruments, tempi, indication de performances.

La frontière entre l'information d'affichage et le contenu musical est parfois poreuse et il revient à l'utilisateur de décider quelle information est en fin de compte pertinente pour lui.

L'objectif de la plateforme NEUMA est de gérer de grandes collections de partitions symboliques, et de répondre aux problématiques majeures du domaine des bibliothèques numériques de partitions (*Digital Score Libraries* ou DSL), telles que la recherche par similarité, le parcours intelligent des documents disponibles, la mise à disposition d'outils d'analyse, ou l'échange et la diffusion de ces documents.

Précisément, on cherche à mettre en place une plateforme qui propose des services de

- enregistrement et stockage de grandes collections, où seule l'information pertinente est conservée (système de *mapping* du contenu musical vers un modèle proposé par la plateforme)
- annotation, partage, modération pour une plateforme collaborative,
- interrogation de plusieurs types, mono ou multi-collections,
- affichage et impression,
- copyright, tatouage et protection des données.

Une fonctionnalité particulière qui distingue NEUMA d'autres types de plateformes est la possibilité de conduire des requêtes portant, indépendamment ou simultanément, sur le texte, le rythme ou la mélodie. Ce type de recherche est sans équivalent dans les domaines autres que musicaux, et n'est pas donc pas implanté sur les plateformes de partages des connaissances type Wikipedia.

On peut imaginer d'autres types de recherche telle que la recherche synchronisée (quel type d'accord est associé au mot « Dieu » dans une collection de musique religieuse ?) ou la recherche d'ancêtre commun (évolution d'un motif musical dans la musique populaire). Ces requêtes dont la traduction à d'autres domaines n'est pas évidente a priori peuvent, une fois disponibles, trouver un sens hors du domaine de recherche d'information musicale.

1.2 Contraintes de la plateforme

L'utilisateur type de la plateforme n'est pas le simple amateur de musique, mais plutôt une institution capable de choisir et produire des collections (ou corpus) de documents, et désireuse d'enrichir la connaissance autour de ces collections par leur étude.

Le contenu publié sera principalement des corpus rares et peu représentés dans le circuit de publication classique, dont la valeur est plus historique que purement esthétique.

Définition 42 (Collection)**.** *On appelle collection de partitions symboliques un ensemble homogène de partitions, c'est-à-dire partageant des caractéristiques communes. Ces caractéristiques peuvent être de nature stylistique, thématique, géographique, historique ou autre.*

Cette définition étant subjective par nature, les outils proposés par le système doivent être suffisamment souples pour adresser une ou plusieurs collections. Par exemple, une collection de chants liturgiques catholiques et une collection de psaumes de la Renaissance ont en commun (outre le fait d'être des chants religieux) la caractéristique d'être une collection de chants monophoniques. Il est donc pertinent de pouvoir appliquer des outils sur l'une ou l'autre des collections, ou sur les deux en même temps. En revanche, une collection de tablatures de guitare a peu de chances d'utiliser les mêmes outils.

Chaque collection peut être décrite par un *modèle*. Un même modèle peut servir à plusieurs collections.

Définition 43 (Modèle)**.** *On appelle modèle une description sémantique du contenu musical.*

Choix d'un modèle. Une partition symbolique contient un grand nombre d'informations. Ces informations peuvent concerner le contenu musical (valeur des notes, durées, instruments, paroles), l'interprétation (tempo, nuances, doigtés) et ou les détails de présentation pour l'affichage final (barres de mesure, titre, armature). Certaines informations sont redondantes, d'autres simplement inutiles.

Il est donc indispensable pour une plateforme fonctionnelle de créer une topographie du contenu pertinent et de l'associer intelligemment à un modèle logique adéquat. Un service permettant d'extraire le contenu pertinent pour l'associer à un modèle correspondant constitue une des bases de la plateforme. Plusieurs modèles logiques peuvent être disponibles sur la plateforme suivant l'évolution du type de partitions enregistrées. Si de nombreuses collections peuvent être traitées avec le modèle présenté en chapitre 2, il ne traite entre autre pas le cas des instruments polyphoniques.

Fonctionnalités. Une fois ce modèle choisi, la plateforme doit proposer une série de fonctionnalités pour la gestion du contenu musical. La plateforme NEUMA se concentre sur les fonctionnalités suivantes :

Recherche par contenu Une fonctionnalité de recherche par contenu s'appuyant sur un modèle défini doit pouvoir exprimer des requêtes complexes sur une ou plusieurs collections de partitions. On résoud ce problème en proposant un modèle unifié et le langage de requête correspondant.

Travail collaboratif, annotations et ontologies Les documents présents sur la plateforme peuvent être enrichis par les utilisateurs, en fonction de leur niveau de connaissance, grâce à un système d'annotations. L'annotation peut être en texte libre ou par termes pré-déterminés dans une ou plusieurs ontologies.

Définition 44 (Ontologie)**.** *Une ontologie est un ensemble structuré de termes et de concepts représentant un domaine d'information. Elle modélise l'ensemble des connaissances d'un domaine.*

Un référentiel global est défini, et renseigné par les musicologues et les utilisateurs. Une annotation consiste en un *tag* laissé par un utilisateur sur un fragment de partition. Le *tag* appartient aux termes de l'ontologie.

Analyse musicale On regroupe derrière ce terme les outils permettant d'explorer le contenu musical autrement que par des requêtes. Ce sujet est celui où l'expertise métier des musicologues intervient le plus.

Copyright et tatouage Un dernier point d'intérêt pour la plateforme est la possibilité de protéger le contenu musical symbolique grâce à un principe de tatouage ou copyright.

1.3 Approche générale

La figure 5.1 met en évidence les différents composants d'un système de gestion de partitions construit autour du modèle de données introduit au chapitre 2. L'architecture est celle d'un SGBD classique, conçue comme une extension du modèle relationnel rendant possible l'intégration de nouvelles fonctionnalités comme composants d'un système extensible.

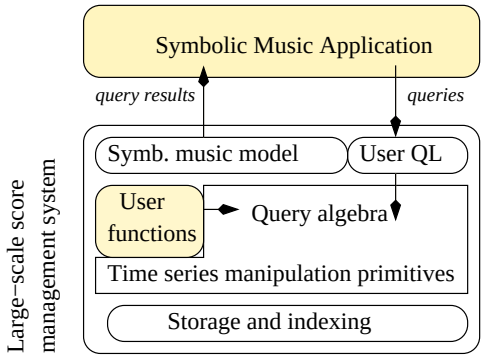


FIG. 5.1 – Détail de l'approche

Le modèle consiste en une vue logique du contenu de la partition, un langage de requête, et une algèbre opérant de manière fermée sur ce modèle (*i.e.* chaque opérateur consomme et produit des instances du modèle). L'algèbre peut être étroitement associée à une bibliothèque de fonctions définies par les utilisateurs, qui permettent d'ajuster le langage de requête aux spécificités du domaine.

Cette approche apporte aux applications spécialisées dans la gestion du contenu musical symbolique les avantages bien connus des SGBD. On rappelle les plus importants : (i) la possibilité de se reposer sur un modèle expressif, stable et bien défini, (ii) l'indépendance entre le modèle logique et la conception physique, ce qui permet d'éviter des problèmes d'optimisation imbriqués au niveau de l'application, et (iii) l'efficacité des opérations ensemblistes et des indexs fournie par le système de données.

2 Description de la plateforme NEUMA

La plateforme NEUMA est conçue pour interagir avec des applications web distantes qui stockent des informations spécifiques à un corpus (fig 5.2).

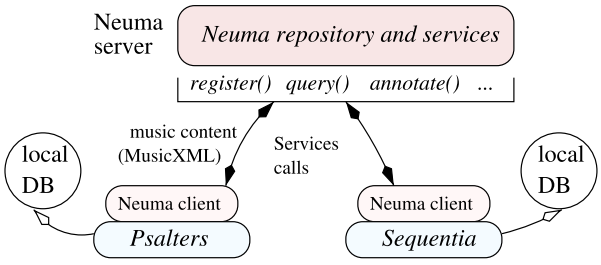


FIG. 5.2 – Bases de données distantes échangeant avec NEUMA

Le but de NEUMA est de gérer tout ce qui a trait au contenu musical, et de laisser les données contextuelles (auteur, compositeur, dates) à l'application cliente.

Une application souhaitant utiliser NEUMA pour la gestion de son contenu musical va faire appel à au moins trois services : *register()* pour envoyer un nouveau document, *query()* pour faire des recherches par contenu, et *render()* pour l'affichage.

2.1 Service *register()*

Pour envoyer un nouveau document, une application appelle le service *register()* (enregistrement). Jusqu'à présent seuls les documents MusicXML sont utilisés pour l'échange de description musicale, mais n'importe quel format peut en principe être utilisé du moment que la fonction de mapping correspondante est mise en place.

Un modèle doit être choisi parmi les modèles proposés. Pour le moment, seul le modèle décrit au chapitre 2 est disponible, et permet de représenter du contenu musical de polyphonie structurée (polyphonie pouvant se ramener à un ensemble de voix structurées). Il n'est pas exclu d'ajouter d'autres types de modèles pour du contenu autre, tels que musique tonale, tablatures de guitare etc. La fonction de mapping extrait du document MusicXML envoyé une représentation du contenu musical conforme à notre modèle. L'ensemble des instances collectées par une application constitue une collection qui sert de base aux autres services. Pendant la phase d'acquisition, une *signature* peut être ajoutée si l'utilisateur veut appliquer un copyright.

2.2 Service *render()*

L'affichage permet à l'utilisateur de visualiser le résultat de sa requête ou de la transformation opérée sur du contenu musical. On affiche l'image d'une partition en appelant le service *render()*. L'image affichée est soit l'image créée à l'enregistrement du contenu, soit une image créée dynamiquement s'il s'agit du résultat d'une transformation. Ce service est basé sur Lilypond[1], programme de gravure de partitions. L'importance d'un modèle unifié apparait clairement dans cet exemple : le service d'affichage est basé sur le modèle, ce qui permet de visualiser facilement une partition transformée. Cette opération serait nettement plus difficile si elle était construite uniquement autour du format du document.

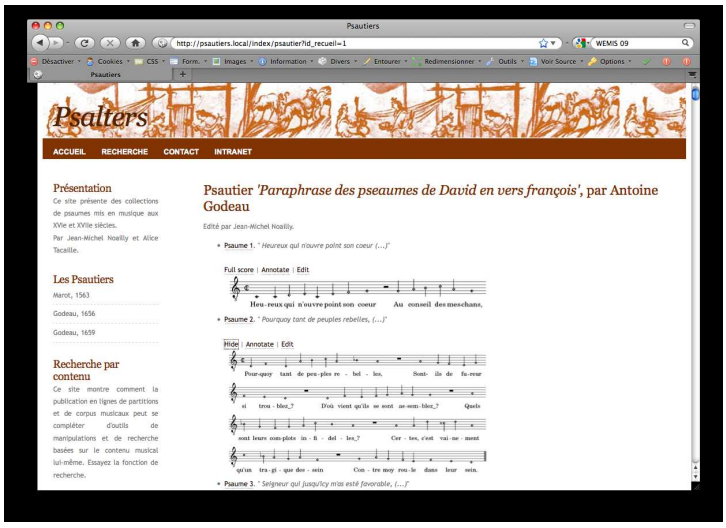


FIG. 5.3 – Affichage

[1] `http://www.lilypond.org`

2.3 Service *query()*

Une grande collection de partitions serait inutile (et inutilisable) si l'on ne disposait pas du service *query()* (recherche) adapté. Ce service doit proposer une recherche par contenu fiable et expressive. Comme il a déjà été dit plus haut, la bibliothèque de partitions numériques NEUMA stocke du contenu musical sans faire de distinction a priori. Par cela on entend que le contenu peut venir de collections diverses, sans rapport les unes avec les autres, et donc potentiellement proposant des partitions de descriptions très hétérogènes. Indépendamment de leurs collections d'origines, le contenu musical est conforme à notre modèle de données, il peut donc être interrogé en conséquence.

Le mécanisme de recherche combine les résultats obtenus dans toutes les collections interrogées.

Différents types de requêtes sont proposés : exacte, transposée, avec ou sans rythme, approchée. Le service de recherche combine la recherche par contenu avec des données de descriptions. Un clavier virtuel est proposé pour entrer le contenu musical.

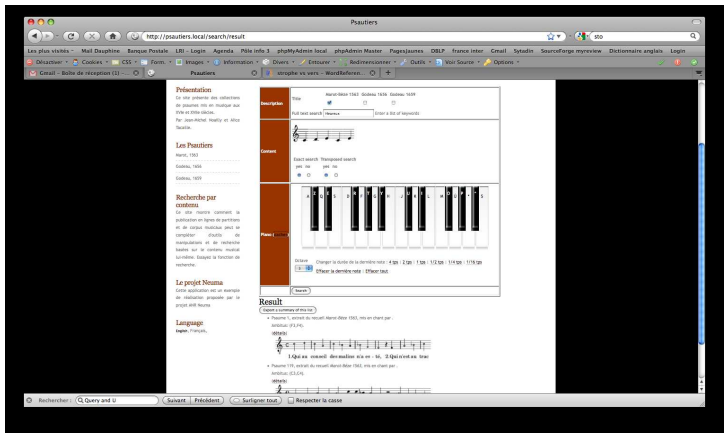


FIG. 5.4 – Saisie de recherche par le clavier virtuel

Le service *query()* supporte aussi le langage utilisateur (cf chapitre 3).

2.4 Service *annotate()*

La plateforme NEUMA fournit également un service d'annotation (*annotate()*). L'annotation est un moyen d'enrichir le contenu symbolique, notamment à des fins de classification. Pour utiliser le service *annotate()*, l'utilisateur sélectionne une portion de partition (ensemble d'éléments de la partition) et saisit les informations concernant cette portion.

Il existe différents types d'annotations : texte libre (utiles pour les indications d'interprétation), ou à choisir parmi les termes pré-déterminés d'une ontologie. Une annotation peut également être une variante de la partie sélectionnée.

Les annotations peuvent être requêtées seules ou en complément des autres critères mentionnés précédemment.

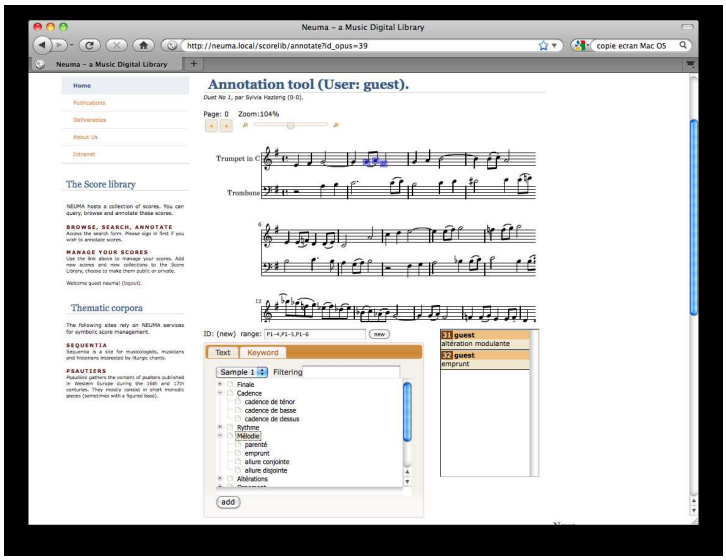


FIG. 5.5 – Annotation par termes pré-déterminés (ontologie)

3 Architecture

Nous choisissons une architecture orientée services (*Service Oriented Architecture* ou SOA).

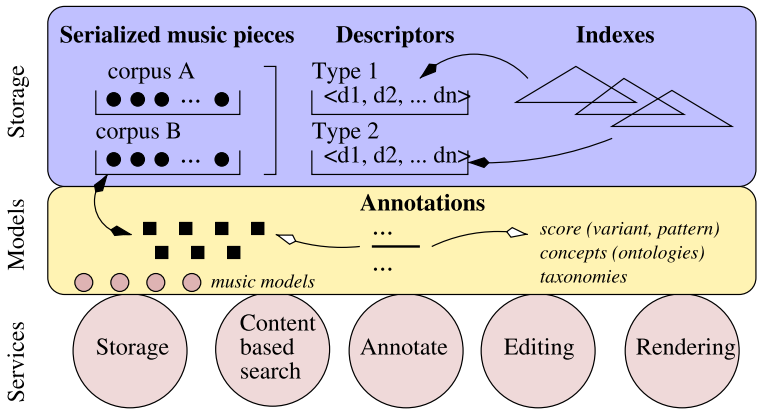


FIG. 5.6 – Architecture du serveur NEUMA

On distingue deux familles de composants : d'un côté les *applications* qui manipulent du contenu musical ; de l'autre la plateforme NEUMA qui consiste en un ensemble de services, chacun destiné à une fonctionnalité spécifique.

On appelle *applications* tout logiciel localisé sur internet qui manipule du contenu musical. Une application communique avec le serveur pour accéder à ses services. NEUMA propose des outils pour faciliter cette communication.

Une application délègue à NEUMA toutes les fonctionnalités liées à la manipulation de contenu, et conserve localement les données contextuelles à sa collection (date de publication, auteurs, compositeurs, lieu de conservation...).

L'architecture de NEUMA est constituée de trois couches :

- la couche de stockage qui gère le stockage et les chemins d'accès,
- la couche logique qui organise l'information en structures haut-niveau,
- la couche de services qui fournit l'interface avec les fonctionnalités.

La couche logique est une pièce centrale du système, qui sert de lien entre les couches haut-niveau et bas-niveau. Elle est constituée d'un ensemble de composant qui définissent la structure des objets manipulés par NEUMA et des opérations qu'ils supportent.

Le contenu musical est une structure qui définit une interprétation d'une pièce de musique. Une interprétation est une abstraction de la partition symbolique. Le contenu jugé inutile, comme les indications de mise en page, est écartée pour cette représentation.

On choisit un modèle M pour un contenu musical spécifique. Quand un nouveau document d est enregistré, le système extrait de la représentation symbolique une instance $i = M(d)$ qui est stockée dans le dépôt NEUMA.

Les instances du modèle consistent en des objet interconnectés, appelé éléments, qui pris tous ensembles décrivent le contenu musical. Chacun de ces éléments a une identité propre. Par exemple une pièce de musique polyphonique, conformément au modèle décrit au chapitre 2, est composée de voix, et chaque voix est une séquence de notes et de silences. Pièce, voix et notes sont des éléments identifiables auquels on peut faire référence à l'intérieur de cette couche logique.

Plus précisément, le contenu musical d'une partition symbolique à polyphonie structurée sera représenté par les modèles MusPiece, MusVoice, MusEvent, MusNote et MusRest (tous des sous-classes successives, sauf pour les deux derniers).

La couche de stockage constitue un dépôt permanent pour d'une part les documents envoyés par l'application qui sont conservés tels quels, et d'autre part la sérialisation des instances du modèle qui constitue la base de données du système.

On extrait des descripteurs de ces instances, c'est-à-dire un encodage sous forme de chaîne de caractères de la séquences des notes et des valeurs rythmiques. Les index pour l'accélération des recherches sont basés sur ces descripteurs. Cette partie est détaillée au chapitre 4.

La troisième couche est l'ensemble des services, déjà abordé partie 2.

4 Conclusion

La mise en place de la plateforme NEUMA montre l'intérêt d'une approche base de données à la gestion des ressources musicales symboliques, et plus spécialement l'utilité d'un modèle unifié pour la représentation du contenu musical.

Les possibilités apportées par une telle plateforme sont loin d'avoir été toutes explorées. Les résultats qu'elle amènera devraient enrichir aussi bien les communautés d'utilisateurs que les acteurs du domaine des bibliothèques numériques.

Conclusion et perspectives

L'étude de la gestion des ressources musicales symboliques soulève de nombreuses questions et implique des domaines de recherches variés.

Pour gérer intelligemment une matière aussi variable et imprévisible que le contenu musical, nous avons choisi une approche algébrique, ce qui a rendu possible la mise en place d'un langage stable et expressif. Nos efforts portent maintenant sur l'optimisation des requêtes.

Il apparait que la démarche proposée dépasse en réalité le domaine de la recherche d'information musicale. De la même manière que les spécificités du domaine musical (comme la synchronisation du texte et de la musique par exemple) ont apporté des pistes de recherche originales, on peut se demander en quoi d'autres domaines manipulant des données temporelles, synchronisées ou non, peuvent bénéficier de cette même approche, voire l'enrichir.

Dans le chapitre 4, nous avons décrit un index pour l'indexation de la recherche exacte et approchée sur du contenu atypique et hétérogène. La méthode que nous proposons est une réponse élégante à ce problème complexe, en particulier car notre réponse prend en considération les différents aspects du contenu musical.

Notre index repose de manière fondamentale sur une notion de signature musicale, variante de la signature algébrique adaptée aux données symboliques. Le terme de *signature musicale* est utilisée en musicologie pour faire référence à un motif musical présent dans plusieurs œuvres d'un compositeur, et qui permet d'en identifier le style [25]. On peut alors s'interroger sur l'interprétation musicale que l'on peut tirer d'une telle signature. Il est envisageable de pouvoir dégager un sens musical si l'on applique ces calculs sur une série de thèmes ou d'incipit. De même, il serait intéressant d'étudier le type de signatures obtenues par compositeurs. Différents paramètres sont alors à prendre en compte, tels que le corps choisi ou la longueur des n-grams.

Plus généralement, l'étude des faux-positifs est intéressante. Un résultat considéré comme faux-positif par un utilisateur non-expert sera peut être pertinent pour un musicologue.

Inversement, dans le cas où la distance A_n est utilisée comme approximation de la distance d'édition (et non comme borne inférieure), on voudrait proposer une caractérisation précise des faux-négatifs. Intuitivement, la répartition des erreurs à l'intérieur du motif semble avoir une influence déterminente. Plus précisément, plus les erreurs sont groupées, plus les deux distances tendent à être les mêmes. Une égalité liant la taille du motif, le nombre d'erreurs tolérées et le n choisi permettrait de caractériser précisément l'approximation faite.

Enfin, d'autres questions restent ouvertes portant à la fois sur les propriétés des biblitohèques numériques et sur les techniques d'indexation.

Comment envisager la recherche par similarité (*i.e.* dans une base de données, trouver les familles de séquences temporelles similaires) ? Une telle recherche peut-elle bénéficier du même type d'index ?

Quels sont les domaines applicatifs autre que la recherche musicale pouvant bénéficier de ce type de travaux ? Les séquences ADN manipulées en bioinformatique et en génomique ne sont pas des séquences temporelles, cependant trouver les extraits de séquences ADN ayant évolué (muté) est une problématique très similaire à la recherche musicale approchée par motif. On peut alors se demander dans quelle mesure les concepts typiquement musicaux comme la recherche transposée peuvent se traduire dans un autre contexte.

Bibliographie

[1] L. Abrouk, H. Audéon, N. Cullot, C. Davy-Rigaux, Z. Faget, D. Gross-Amblard, H. Lee, P. Rigaux, A. Tacaille, E. Gavignet, and V. Thion-Goasdoué. The neuma project : Towards cooperative on-line music score libraries. In *Workshop on Exploring Musical Information Spaces (WEMIS)*, 2009.

[2] N. Adams, M. Bartsch, J. Shilfrin, and G. Wakefield. Time series alignment for music information retrieval. In *Proc. Intl. Society for Music Information Retrieval (ISMIR)*, 2004.

[3] N. H. Adams, M. A. Bartsch, J. B. Shifrin, and G. H. Wakefield. Time series alignment for music information retrieval. In *Proc. Intl. Society for Music Information Retrieval (ISMIR)*, 2004.

[4] M. Agosti, H. Albrechtsen, N. Ferro, I. Frommholz, P. Hansen, N. Orio, E. Panizzi, A. M. Pejtersen, and U. Thiel. DiLAS : a Digital Library Annotation Service. In *Proc. Annotation for Collaboration*, pages 91–101, 2005.

[5] M. Agosti and N. Ferro. A system architecture as a support to a flexible annotation service. In *DELOS Workshop : Digital Library Architectures*, pages 147–166, 2004.

[6] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 1990.

[7] C. Alzola and F. Harell. An introduction to s-plus and the hmisc and design libraries, 1999. Online version at *http ://fharrell.biostat.virginia.edu/s/index.html*.

[8] A. Bahri, Y. Naïja, M. Manouvrier, and G. Jomier. Recherche par similarité de séquences : un état de l'art. In *Manifestation de Jeunes Chercheurs STIC*, 2004.

[9] D. Bainbridge, C. G. Nevill-Manning, I. H. Witten, L. A. Smith, and R. J. McNab. Towards a digital library of popular music. In *Proc. of the Int. Conf. on Digital Libraries (ICDL)*, pages 161–169, 1999.

[10] J. Band. The google library project, both sides of the story. In *Plagiary Cross Disciplinary Studies in Plagiarism, Fabrication, and Falsification*, page 117, 2006.

[11] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Workshop on Knowledge Discovery in Databases*, pages 229–248, 1994.

[12] C. Borgman. What are digital libraries ? competing visions. In *Information Processing and Management (35)*, pages 227–243, 1999.

[13] D. Byrd. Music-notation searching and digital libraries. In *Proc. of ACM/IEEE-CS Joint Conf. on Digital Libraries (JCDL)*, pages 239–246, 2001.

[14] X. Cao, S. C. Li, and A. K. H. Tung. Indexing dna sequences using q-grams. In *Proc. Int. Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 4–16, 2005.

[15] E. Cesar and B. Richard. *Les Séries Temporelles.* Cours de l'Université de Versailles Saint Quentin, 2006. Online version at *http ://-georges.gardarin.free.fr/Surveys_ DM/Survey_ Time_ Series.pdf.*

[16] A. Charpentier. *Time series : theory and applications.* Cours de l'Université Paris Dauphine, 2005. Online version at *http ://www.math.univ-montp2.fr/ ribereau/matos/TS1.pdf.*

[17] B. Chiu, E. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. In *Proc. of Int. Conf. on Knowledge Discovery and Data Mining (SIGKKD)*, pages 493–498, 2003.

[18] J. Chomicki. Temporal query languages : A survey. In *Int. Conf. Temporal Logic (ICTl)*, pages 506–534, 1994.

[19] S. Chu, E. Keogh, D. Hart, and M. Pazzani. Iterative deepening dynamic time warping. In *SIAM Int. Conf. on Data Mining (SDM)*, 2002.

[20] J. Clifford and A. U. Tansel. On an algebra for historical relational databases : Two views. In *Special Interest Group on Management of Data Conf. (SIGMOD)*, pages 247–265, 1985.

[21] D. Conklin. Representation and discovery of vertical patterns in music. In *Music and Artificial Intelligence : Lecture Notes in Artificial Intelligence*, number 2445, page 32–42, 2002.

[22] D. Conklin and M. Bergeron. Discovery of contrapuntal patterns. In *Proc. Intl. Society for Music Information Retrieval (ISMIR)*, pages 201–206, 2010.

[23] C. Constantin, Z. Faget, C. du Mouza, and P. Rigaux. Indexing symbolic music scores. In *Bases de Données Avancées (BDA)*, 2011.

[24] C. Constantin, Z. Faget, C. du Mouza, and P. Rigaux. The melodic signature index for fast content-based retrieval of symbolic scores. In *Proc. Intl. Society for Music Information Retrieval (ISMIR)*, 2011.

[25] D. Cope. *Experiments in Music Intelligence.* A-R Editions, 1996.

[26] T. Crawford, C. S. Iliopoulos, and R. Raman. String-matching techniques for musical similarity and melodic recognition. *Computing in Musicology*, 11 :71–100, 1998.

[27] S. Doraisamy and S. M. Rüger. An approach towards a polyphonic music retrieval system. In *Proc. Intl. Society for Music Information Retrieval (ISMIR)*, pages 187–93, 2001.

[28] J. S. Downie. Music information retrieval (chapter 7). In *Annual Review of Information Science and Technology (37)*, pages 295–340, 2003.

[29] S. Downie and M. Nelson. Evaluation of a simple and effective music information retrieval method. In *Special Interest Group on Information Retrieval (SIGIR)*, pages 73–80, 2000.

[30] W. Dreyer, A. K. Dittrich, and D. Schmidt. An object-oriented data model for a time series management system. In *Int. Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 186–195, 1994.

[31] C. du Mouza, W. Litwin, P. Rigaux, and T. J. E. Schwarz. AS-index : a Structure for String Search Using n-grams and Algebraic Signatures. In *Proc. Intl. Conf. on Information and Knowledge Management (CIKM)*, pages 295–304, 2009.

[32] F. Duchene, C. Garbay, and V. Rialle. Similarity measure for heterogeneous multivariate time-series. In *Proc. of European Signal Processing Conference (EUSIPCO)*, 2004.

[33] D. Eppstein. Design and analysis of algorithms, 1996. Online version at *http ://www.ics.uci.edu/ eppstein.*

[34] Z. Faget and P. Rigaux. A database approach to symbolic music content management. In *Int. Symp. on Computer Music Modeling and Retrieval (CMMR)*, pages 303–320, 2010.

[35] Z. Faget, P. Rigaux, D. Gross-Amblard, and V. Thion-Goasdoué. Modeling synchronized time series. In *Int. Database Engineering and Applications Symp. (IDEAS)*, pages 82–89, 2010.

[36] C. Faloutosos, R. Agrawal, and A. Swami. Efficient similarity search in sequence databases. In *Proc. of Int. Conf. on Foundations of Data Organizations and Algorithms (FODO)*, pages 1–15, 1993.

[37] M. Fingerhut. Music information retrieval, or how to search for (and maybe find) music and do away with incipits. In *Assoc. Int. des Archives Sonores et Audiovisuelles(IASA) and Int. Assoc. of Music Libraries, Archives, and Documentation Centres (IAML) Congress*, 2004.

[38] G. Groves and E. J. Hannan. Time series regression of sea level on weather. *Rev. Geophysics*, page 129–174, 1968.

[39] J. Hamilton. *Time Series Analysis.* Princeton, 1994.

[40] R. Hillewaere, B. Manderick, and D. Conklin. String quartet classification with monophonic models. In *Proc. Intl. Society for Music Information Retrieval (ISMIR)*, 2010.

[41] C. Jensen, J. Clifford, S. Gadia, A. Segev, and R. Snodgrass. A glossary of temporal database concepts. *Sigmod Record*, 21, 1992.

[42] C. S. Jensen and R. T. Snodgrass. Temporal data management. In *IEEE Trans. Knowl. Data Eng.*, volume 11, pages 36–44, 1999.

[43] J. Keiper, H. Brocks, A. Dirsch-Weigand, A. Stein, and U. Thiel. Collate - a web-based collaboratory for content-based access to and work with digitized cultural material. In *Int. Cultural Heritage Informatics Meeting (ICHIM)*, pages 495–511, 2001.

[44] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIG-MOD Record (ACM Special Interest Group on Management of Data)*, volume 30 (2), page 151–162, 2001.

[45] E. Keogh and M. Pazzani. Relevance feedback retrieval of time series data. In *Proc. of Int ACM-SIGIR Conf. on Research and Development in Information Retrieval*, pages 183–190, 1999.

[46] E. Keogh and M. Pazzani. Dynamic time warping with higher order features. In *SIAM Int. Conf. on Data Mining (SDM)*, 2001.

[47] E. J. Keogh and C. A. Ratanamahatana. Exact Indexing of Dynamic Time Warping. In *Knowl. Inf. Syst.*, volume 7, pages 358–386, 2005.

[48] B. Kucera. On the periodicity of sunspots. *DML-CZ*, 35 :172–179, 1906.

[49] O. Lartillot. Pattern mining in discrete time series and application to music mining. In *Proc. of Conf. on Advances in Intelligent IT : Active Media Technology (AMT)*, pages 144–149, 2006.

[50] J. Lee and J. S. Downie. Survey of music information needs, uses, and seeking behaviours : Preliminary findings. In *Proc. Intl. Society for Music Information Retrieval (ISMIR)*, 2004.

[51] J. Lee and R. Elmasri. An eer-based conceptual model and query language for time-series data. In *Proc. Intl. Conf. on Conceptual Modeling*, pages 21–34, 1998.

[52] A. Lerner and D. Shasha. AQuery : Query language for ordered data, optimization techniques and experiments. In *Int. Conf. on Very Large Data Bases (VLDB)*, 2003.

[53] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10 :707, 1966.

[54] W. Litwin, R. Mokadem, P. Rigaux, and T. Schwarz. Fast nGram Based String Search over Data Encoded Using Algebraic Signatures. In *Proc. Intl. conf. on Very Large Database (VLDB)*, 2007.

[55] H. Lütkepohl. *A New Introduction to Time Series Analysis*. Springer, 2005.

[56] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. Online version at *http ://informationretrieval.org/*.

[57] A. Mueen and E. Keogh. Online discovery and maintenance of time series motifs. In *Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 2010.

[58] G. Navarro, R. Baeza-yates, E. Sutinen, and J. Tarhio. Indexing Methods for Approximate String Matching. *IEEE Data Engineering Bulletin*, 24(4) :19–27, 2001.

[59] T. Negi and V. Bansal. Time series : Similarity search and its applications. In *Int. Conf. Systemics, Cybernetics and Informatics (ICSCI)*, pages 528–533, 2005.

[60] NIST-SEMATEXH. Nist/sematech e-handbook of statistical methods, 2010. Online version at *http ://www.itl.nist.gov/div898/handbook*.

[61] P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *Intl. Conf. on Data Mining (ICDM)*, 2002.

[62] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. In *Proc. of the National Academy of Sciences of the USA*, volume 85, pages 2444–2448, 1988.

[63] D. Rafiei and A. O. Mendelzon. Querying time series data based on similarity. *Trans. on Knowledge and Data Engineering*, 12 :675–693, 2000.

[64] M. Ranganathan, C. Faloutosos, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. of Conf. on Management of Data*, 1994.

[65] S. O. Rice. Noise in fm receivers. *Time Series Analysis*, 1963.

[66] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. In *IEEE Trans. Acoustics, Speech and Signal Proc*, volume ASSP-26, pages 43–49, 1978.

[67] T. Saracevic and M. Dalbello. A survey of digital library education. In *Proc. of the American Society for Information Science and Technology (ASIST)*, pages 209–223, 2001.

[68] E. Schubert. Analysis of emotional dimensions in music using time series techniques. In *Jour. Music Research*, volume 31, pages 65–80, 2007.

[69] A. Segev and R. Chandra. A data model for time-series analysis. In *Advanced Database Systems*, pages 191–212, 1993.

[70] A. Segev and A. Shoshani. Logical modeling of temporal data. In *Special Interest Group on Management of Data Conf. (SIGMOD)*, pages 454–466, 1987.

[71] A. Segev and A. Shoshani. The representation of a temporal data model in the relational environment. In *Int. Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 39–61, 1988.

[72] D. Shasha. Time series in finance : The array database approach, 1998. Online version at *http ://cs.nyu.edu/shasha/papers/jagtalk.html*.

[73] D. Shasha and A. Lerner. Aquery : query language for ordered data, optimization techniques, and experiments. In *Int. Conf. on Very Large Data Bases (VLDB)*, pages 345 – 356, 2003.

[74] J. Shieh and E. J. Keogh. isax : indexing and mining terabyte sized time series. In *Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 623–631. ACM, 2008.

[75] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147 :195–197, 1981.

[76] R. B. Stam and R. T. Snodgrass. A bibliography on temporal databases. In *IEEE Data Eng. Bull.*, volume 11, pages 53–61, 1988.

[77] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass. *Temporal Databases : Theory, Design, and Implementation*. Database Systems and Applications Series, Benjamin/Cummings Pub. Co., Redwood City, CA, 1993.

[78] R. Typke, P. Giannopoulos, R. C. Veltkamp, F. Wiering, and R. V. Oostrum. Using transportation distances for measuring melodic similarity. In *Proc. Intl. Society for Music Information Retrieval (ISMIR)*, pages 107–114, 2003.

[79] R. Typke, F. Wiering, and R. Veltkamp. A survey of music information retrieval systems. In *Proc. Intl. Society for Music Information Retrieval (ISMIR)*, 2005.

[80] A. Uitdenbogerd and J. Zobel. Matching techniques for large music databases. In *Proc. ACM Multimedia Conference (ACMC)*, pages 57–66, 1999.

[81] E. Ukkonen. Approximate String Matching with q-grams and Maximal Matches. *Theoretical Computer Science*, 92 :191–211, 1992.

[82] P. E. Utgoff and P. B. Kirlin. Detecting motives and recurring patterns in polyphonic music. In *Int. Computer Music Conference (ICMC)*, page 487–494, 2006.

[83] M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In *Proc. Intl. Conf. on Data Engineering (ICDE)*, pages 673–684, 2002.

[84] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes : Compressing and Indexing Documents and Images*. Morgan-Kaufmann, 1999.

[85] Y.-D. Wu, Y. Li, and B.-L. Liu. A new method for approximate melody matching. In *Int. Conf. on Machine Learning and Cybernetics*, 2003.

[86] T. Yuzuriha. The autocorrelation curves of schizophrenic brain waves and the power spectrum. *Psychiatra and Neurologica Japonica*, pages 911–924, 1960.