Estimation du contrôle territorial dans le cadre de conflits insurrectionnels à l'aide de chaînes de Markov cachées

Camille Ballu Jacques-Yves Guilbert--Ly

28 mai 2021

Introduction

Nous avons fait le choix pour ce projet informatique de travailler sur un sujet de recherche porté sur une problématique géopolitique. Cette thématique, de même que le choix du groupe, ont été motivés par une volonté de s'atteler à quelque chose d'intrinsèquement lié à nos aspirations professionnelles communes. Après avoir étudié pendant quelques semaines la littérature scientifique du domaine de la géopolitique et des relations internationales, nous avons retenu l'application des chaînes de Markov cachées au domaine de l'occupation territoriale, développée par Therese Anders dans un article de 2019, *Territorial control in civil wars : Theory and measurement using Machine Learning* [1] comme un exemple innovant et stimulant à analyser dans le cadre d'un projet informatique. C'est un thématique d'autant plus intéressante à nos yeux que la littérature à ce sujet est assez peu développée, et qu'elle nous a donc invités à prendre considérablement des initiatives en terme de codage.

Les chaînes de Markov cachées sont une complexification des chaînes de Markov classiques qui, elles, visent à définir des probabilités pour des séquences d'événements observés. Les chaînes de Markov cachées fonctionnent sur un principe similaire, mais s'intéressent à des événements cachés derrière des événements observables. Elles ont des applications nombreuses et très variées dans le domaine de la Data Science pour ce qui concerne par exemple les processus de reconnaissance (de parole, de voix, d'écriture manuscrite) ou encore les processus de prédiction (météorologique par exemple, ou en bio-informatique avec la prédiction des gènes). Ici, on va chercher à les appliquer à l'analyse du contrôle territorial des zones de conflits entre un État et des institutions rebelles à partir de l'observation des types de combats qui se déroulent sur le territoire.

Le fait d'utiliser les chaînes de Markov cachées dans le cadre de cette problématique apparaît particulièrement approprié dans la mesure où les organismes internationaux, concernés par des problématiques économiques et sociales, manquent bien souvent de données empiriques sur l'occupation des territoires dans les zones de guerre civiles, que ce soit parce que ce sont des zones enclavées mais aussi parce que cette occupation est la plupart du temps très volatile, et ce tout particulièrement pour les combats asymétriques. Il est essentiel par exemple pour des ONG de connaître quel type d'acteur contrôle un territoire pour venir aider les populations en détresse. Les chaînes de Markov cachées permettent d'offrir une estimation du contrôle territorial de ces zones contestées à partir des types de combats et d'attaques qui s'y produisent, allant des actes terroristes à des techniques de combats plus conventionnelles.

Nous avons donc construit un modèle reliant la fréquence relative des actes terroristes et des actes de guerre conventionnels à des schémas de contrôle territorial. Pour cela, nous avons admis deux relations empiriques essentielles : les rebelles utilisent principalement les méthodes terroristes au-delà des territoires qu'ils possèdent et préfèrent les méthodes de guerre conventionnels lorsque cela concerne des territoires où ils ont une forte influence.

Pour notre étude, nous nous sommes appuyés sur deux bases de données. La première est la *Global Terrorism Database* (GTD) qui recense les incidents terroristes depuis 1970 ; elle est maintenue par le *National Consortium for Study of Terrorism and Responses to Terrorism* (START) de l'Université du Maryland et contient plus de 190 000 entrées. La seconde base de données est celle de *Upssala Conflict Data Program* de l'Université d'Uppsala en Suède ; elle catalogue un ensemble d'évènements de violence armée, notamment au sein de guerres civiles. La base de données est mise à jour annuellement, à l'occasion d'une publication dans le *Journal of Peace Research*.

1 La problématique du contrôle territorial

Afin de bien saisir les enjeux de ce qui va être modélisé dans ce projet, il est nécessaire de revenir dans un premier temps sur la problématique du contrôle territorial pour s'assurer d'en comprendre les tenants et les aboutissants, et ce tout particulièrement pour l'Irak et le Nigeria.

Le contrôle territorial est l'un des enjeux majeurs des zones de conflits entre un état contesté et des pouvoirs rebelles puissants. Le gain de territoires est un objectif important pour les belligérants qui s'opposent dans le cadre des conflits asymétriques, pour les forces rebelles, pour qui cela signifie souvent un gain de puissance, comme pour le gouvernement. En effet, le contrôle d'un territoire permet par exemple d'en exploiter les ressources naturelles et humaines. La prise en main un territoire est la plupart du temps synonyme d'une baisse des combats dans la zone concernée, entre autres dès lors qu'un gouvernement politique clairement défini s'y est établi. Elle est aussi surtout synonyme d'une baisse de la violence indiscriminée envers les populations civiles de la part du pouvoir qui possède le territoire. Dans l'ensemble, moins les acteurs d'un conflit contrôlent un territoire, plus la violence qu'ils infligeront sera aveugle, et vice versa.

Polo et Gleditsch [2] affirment que le terrorisme diffère des attaques conventionnelles dans les conflits civils en ce que les cibles ou les victimes immédiates ne sont généralement pas des combattants, et en ce que chaque victime individuelle est souvent moins importante que l'objectif de transmettre un message au public visé. L'absence de contrôle territorial est un facteur clé pour expliquer l'utilisation du terrorisme par les insurgés par opposition aux tactiques de guérilla plus conventionnelles : le recours au terrorisme est marqueur de l'incapacité des rebelles à contrôler un territoire. Nous sommes partis de ce constat pour implémenter notre algorithme : des niveaux plus élevés de contrôle territorial des rebelles sont associés à des niveaux plus élevés de combats conventionnels tandis que des niveaux plus élevés de contrôle gouvernemental sont associés à davantage de terrorisme. Le contrôle territorial par des rebelles se manifeste davantage par des tactiques de guérilla telles que les attaques à la fuite, les embuscades, les raids et les batailles à petite échelle.

Pour cette étude, nous nous intéressons à deux pays en particulier : le Nigéria, et l'Irak. Le Nigéria connaît depuis le mois de juillet 2009 une insurrection initiée par le groupe djihadiste Boko Haram. Si le soulèvement a été tout d'abord contenu par le gouvernement nigérian, le groupe n'a pas été entièrement vaincu. L'organisation, menée par Abubakar Shekau, a entrepris une campagne terroriste dans les années qui ont suivi l'insurrection de 2009 et a monté en puissance, si bien qu'elle est parvenue progressivement à s'emparer des villes au Nord-Est du pays à partir de 2012. Cette montée en puissance, qui culmine en 2014, s'est accompagnée d'une militarisation des moyens du groupe, qui a reçu notamment le soutien de l'Etat Islamique en Irak et au Levant. Une réponse contre-insurrectionnelle internationale a été apportée par la Force multinationale mixte en 2015, une coalition composée du Tchad, du Niger, du Nigéria, du Bénin et du Cameroun. Le groupe djihadiste a été progressivement repoussé vers le lac Tchad où il a établi son sanctuaire. Boko Haram, ainsi que l'État islamique en Afrique de l'Ouest (ISWAP) qui a été formé suite à une scission avec le premier, continuent en 2021 à poser une menace sécuritaire régionale majeure.

L'Irak est un pays à situation sécuritaire précaire depuis l'invasion américaine en 2003. Le renversement de Saddam Hussein et les politiques post-bellum de reconstruction ont posé le terreau pour l'émergence d'un insurrection anti-gouvernementale alimentée par des griefs sectaires et religieux. Se sont ensuit sept ans de combats contre-insurrectionnels à travers un pays plongé dans une guerre civile. En 2011, les insurgés ont été relativement battus par le gouvernement irakien et la coalition menée par les Américains. Cependant, le retrait du pays des troupes américaines au moment d'un aggravement de la situation sécuritaire en Syrie lié à la révolution, a favorisé grandement la résurgence de groupes djihadiste à travers le pays. Le plus notable est assurément l'Etat Islamique (EI), qui est parvenu progressivement à partir de 2013 à s'emparer de l'Est du pays. Alors que la situation devenait catastrophique, l'EI se rapprochant dangereusement de la capitale, l'Irak a décidé en 2014 d'appeler à l'aide la communauté internationale ; une coalition menée par les Etats-Unis est intervenue pour fournir un soutien militaire important aux troupes irakiennes et kurdes. Elles sont parvenues ensemble à repousser l'EI, qui a été déclaré vaincu militairement en 2017. Les violences armées et attentats n'ont pour autant pas cessé depuis.

Maintenant que nous avons introduit le contexte géopolitique qui encadre notre projet, nous allons pouvoir présenter le principe des chaînes de Markov et des chaînes de Markov cachées en vue de construire notre modèle.

2 Les chaînes de Markov cachées

2.1 Les chaînes de Markov

Une chaîne de Markov est un modèle probabiliste qui donne des informations sur les probabilités de séquences de variables aléatoires, appelées états, prenant des valeurs dans un ensemble quelconque (des mots, des symboles, la météo, ou encore des états comme malade/sain, dangereux/sécurisé....). Ici nous allons nous intéresser en guise d'états au différents degrés de contrôle territorial d'une zone. Une chaîne de Markov fait l'hypothèse très forte que si nous voulons prédire ce qu'il va advenir dans la séquence, tout ce qu'il faut prendre en compte est l'état actuel. Les états antérieurs à l'état actuel n'ont aucun impact sur l'avenir si ce n'est via l'état actuel qu'ils ont engendré.

Afin de bien comprendre le fonctionnement des chaînes de markov cachées, nous avons fait le choix d'implémenter un exemple purement explicatif du passage des chaînes de Markov classiques aux chaînes de Markov cachées dans le document `exemple_hmm.py` Regardons dans un premier temps l'implémentation d'une chaîne de Markov classique, et, pour rester dans la thématique des conflits asymétriques, considérons la nature des occupants sur un territoire contesté. On leur attribue (de façon arbitraire car cela est fait à titre d'exemple) une probabilité d'occupation initiale respective de 0,6 et de 0,4. C'est la probabilité que la chaîne de Markov commence à l'état « rebelles »ou à l'état « gouvernement ». On utilise le module pandas pour constituer un tableau.

```
states = ['rebelles', 'gouvernement']
pi = [0.6, 0.4]
```

Nous définissons ensuite la matrice stochastique a de transition d'un état à l'autre en utilisant la fonction DataFrame du module pandas, encore une fois de façon arbitraire.

```
a_df = pd.DataFrame(columns=states, index=states)
a_df.loc[states[0]] = [0.7, 0.3]
a_df.loc[states[1]] = [0.4, 0.6]
a = a_df.values
```

C'est à partir de cette matrice stochastique que l'on peut réaliser des récurrences sur les variables aléatoires représentant les états de la chaîne de Markov (rebelles et gouvernement). On peut tracer le graphe probabiliste correspondant à la situation à l'aide du module Graphviz (Figure 1).

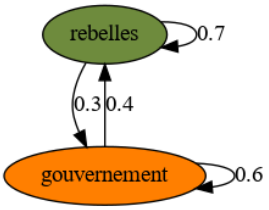


Figure 1 – Tactiques & contrôle territorial

2.1.1 Les chaînes de Markov cachées

Passons maintenant aux chaînes de Markov cachées. Une chaîne Markov est utile lorsqu'il est nécessaire de calculer une probabilité pour une séquence d'événements observables. Dans de nombreux cas, cependant, les événements qui nous intéressent sont cachés et ne sont pas observables directement. Une chaîne de Markov cachée va nous permettre de parler à la fois d'événements observés et d'événements cachés. Par exemple, les chaînes de Markov cachées peuvent aider à déterminer la probabilité qu'une personne soit malade ou saine (états cachés) en fonction de son comportement (états observables). Les différents éléments qui composent les chaînes de Markov cachées sont donnés en Table 1.

$S = \{s_0, \dots, s_{N-1}\}$	un ensemble de N états
$T = (t_{ij}) \in \mathcal{M}_N([0,1])$	une matrice de probabilité de transition : t_{ij} représente la probabilité de passer de l'état s_i à l'état s_j
$O = (o_0, \dots, o_{T-1})$	une séquence de T observations
$E = (e_{it}) \in \mathcal{M}_{N,T}([0,1])$	une matrice probabilités d'émission : e_{it} est la probabilité que l'observation o_t soit générée par l'état s_i
$\Pi = (\pi_0, \dots, \pi_{N-1})$	les probabilités initiales de chaque état : π_i est la probabilité que la chaîne de Markov commence à l'état s_i

Table 1 – Les composants d'une chaîne de Markov cachée – Jurafsky & Martin, 2021 [3]

Comme pour les chaînes de Markov classiques, chaque événement d'une chaîne de Markov cachée dépend de l'état qui l'a précédé et la probabilité d'une observation dépend uniquement de l'état qui a produit l'observation, et non pas d'un autre état ou d'une autre observation (par indépendance).

Nous allons illustrer une fois encore par un exemple explicatif le processus des chaînes de Markov cachées dans le document `exemple_hmm.py`. Les états observables restent les mêmes et nous définissons trois états cachés derrière ces événements observables : 'état', 'neutre' et 'rebelle'. Nous définissons enfin les probabilités initiales de chaque état caché.

```
observable_states= ['attentats', 'guerillas']
hidden_states = ['etat', 'neutre', 'rebelle']
pi = [0.5, 0.1, 0.4]
state_space = pd.Series(pi, index=hidden_states, name='states')
```

L'enjeu est ensuite de définir les matrices de transition et d'émission de la chaîne de Markov cachée (ici de façon aléatoire pour illustrer l'exemple). C'est l'existence de ces deux matrices qui constitue toute la spécificité des chaînes de Markov cachées vis-à-vis des chaînes de Markov classiques.

```
# on définit t la matrice de transition (probabilité de changer d'état en fonction de chaque
↪  état)

t_df = pd.DataFrame(columns=hidden_states, index=hidden_states)
t_df.loc[hidden_states[0]] = [0.6, 0.250, 0.150]
t_df.loc[hidden_states[1]] = [0.350, 0.2, 0.450]
t_df.loc[hidden_states[2]] = [0.3, 0.2, 0.5]

t = t_df.to_numpy(dtype='float64')

# on definit la matrice d'emission (probabilite de realisation d'un etat en fonction de
↪  l'observation)

e_df = pd.DataFrame(columns=observable_states, index=hidden_states)
e_df.loc[hidden_states[0]] = [0.8, 0.2]
e_df.loc[hidden_states[1]] = [0.6, 0.4]
e_df.loc[hidden_states[2]] = [0.3, 0.7]

e = e_df.to_numpy(dtype='float64')
```

Enfin, on peut illustrer comme précédemment l'exemple par un graphe probabiliste à l'aide du module graphviz (Figure 2).

Un article influent de Rabiner [4], a introduit l'idée que la plupart des modèles de Markov cachés pouvait se regrouper en trois problèmes fondamentaux, décrits en Table 2.

Ce projet va appliquer la deuxième méthode présentée, celle du décodage, à la question du contrôle des zones de conflit au sein des territoires contestés. C'est la technique qu'emploie notamment Thérèse Anders dans son article. Nous nous en sommes inspirés, et nous avons adapté son travail effectué sous R concernant le Nigeria pour étudier le cas à la fois de l'Irak et du Nigeria. La finalité étant de parvenir à produire une carte du contrôle rebelle et étatique des deux pays, sur une période allant de 2008 à 2019

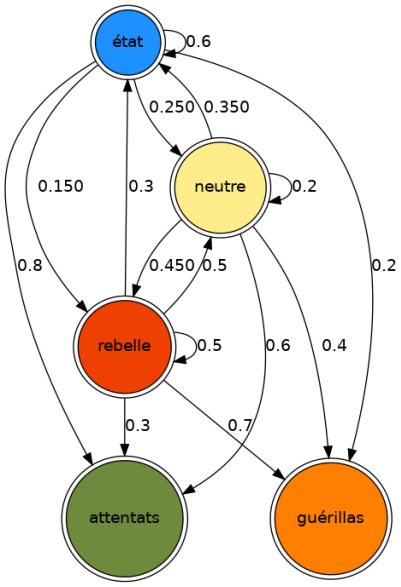


FIGURE 2 – Tactiques & contrôle territorial

Prob. 1 (Probabilité)	Etant donné un MCC λ et une séquence d'observation O, calculer la probabilité de cette séquence $P_\lambda(O)$
Prob. 2 (Décodage)	Etant donné un MCC λ et une séquence d'observation O, déterminer la meilleure séquence d'états $(q_i) \in Q^T$.
Prob. 3 (Apprentissage)	Etant donné une séquence d'observation O et un ensemble d'états, déterminer les paramètres T et E du MCC

TABLE 2 – Trois problèmes caractérisants les modèles de Markov cachés

pour le Nigéria, et de 2011 à 2019 pour l'Irak. Pour parvenir à cette finalité, nous avons dû dans un premier temps récolter et exploiter les données nécessaires à l'analyse.

3 Détermination des séquences d'observations

La première étape de la modélisation consiste à définir les séquences d'observations sur lesquelles nous appliquerons un algorithme de décodage. Afin de déterminer le contrôle territorial du pays, nous subdivisons ce pays en cellules hexagonales que nous traitons individuellement. L'objectif est de déterminer pour chacune de ces cellules, le niveau de contrôle de cette cellule, au cours d'une période donnée. Nous modélisons le contrôle territorial par une variable pouvant prendre cinq valeurs, détaillées en Table 3.

Variable	Signification	Description
R	*Rebel*	Territoire sous contrôle rebelle
DR	*Disputed Rebel*	Territoire contesté, plutôt sous contrôle rebelle
D	*Disputed*	Territoire contesté
DG	*Disputed Government*	Territoire contesté, plutôt sous contrôle gouvernemental
G	*Government*	Territoire sous contrôle gouvernemental

TABLE 3 – États de contrôle territorial – Anders, 2020 [1]

Le contrôle territorial définit l'état d'une cellule ; c'est cette séquence d'état que nous cherchons à déterminer à partir d'une séquence d'observation.

La séquence d'observation d'une cellule est inférée à partir d'une mesure de l'exposition de la cellule à des évènements terroristes ou de combats conventionnels (guérilla). Étant donné deux mesures E^T_{it} et E^C_{it} de l'exposition d'une cellule i à la date t aux terrorisme et aux combats conventionnel, nous déduisons deux

probabilités d'exposition $T_{it} = P_{\lambda_t^T}(E_{it}^T)$ et $C_{it} = P_{\lambda_t^C}(E_{it}^C)$ au terrorisme et aux combats conventionnels. Ici, P_λ suit une loi de Poisson de paramètre λ ; et λ_t^T et λ_t^C désignent respectivement les expositions moyennes de l'ensemble des cellules aux terrorisme et combats conventionnels pendant le mois t. Ces probabilités nous permettent de déduire, pour chaque cellule i et chaque date t, une observation à partir des modalités de combat, comme détaillé en Table 4.

Observation	Tactique	Description	Commentaires
$o_{it} = O_1$	$E_{it}^T \approx E_{it}^C$	Peu ou pas d'évènements terroristes ou conventionnels.	Les expositions observées inférieures à un seuil $s = 0.01$ sont tronquées à zéro.
$o_{it} = O_2$	$C_{it} > T_{it}$ et $\lvert C_{it} - T_{it}\rvert > m$	Exposition supérieure au combats conventionnels qu'au terrorisme.	
$o_{it} = O_3$	$\lvert C_{it} - T_{it}\rvert \leq m$	Exposition similaire aux terrorisme et combats conventionnels.	m contrôle l'intervalle de superposition des probabilités.
$o_{it} = O_4$	$C_{it} < T_{it}$ et $\lvert C_{it} - T_{it}\rvert > m$	Exposition supérieure aux combats conventionnel qu'au terrorisme.	

Table 4 – Codage des observations – Anders, 2020 [1]

Les zones à exposition relativement faible à la violence armée sont supposées être sous contrôle total du gouvernement ou des groupes rebelles Les zones plus exposées au terrorisme sont supposées être plutôt sous un contrôle gouvernemental contesté. A l'inverse, les zones plus exposées aux combats conventionnels sont supposées être plutôt sous un contrôle insurgé La présence en proportion similaire de chacune des deux tactiques dans une zone est caractéristique d'une zone très disputée. Le tout est résumé en Figure 3.

Ce sont à ces séquences d'observation que nous allons appliquer l'algorithme de Viterbi, présenté dans la partie qui suit.

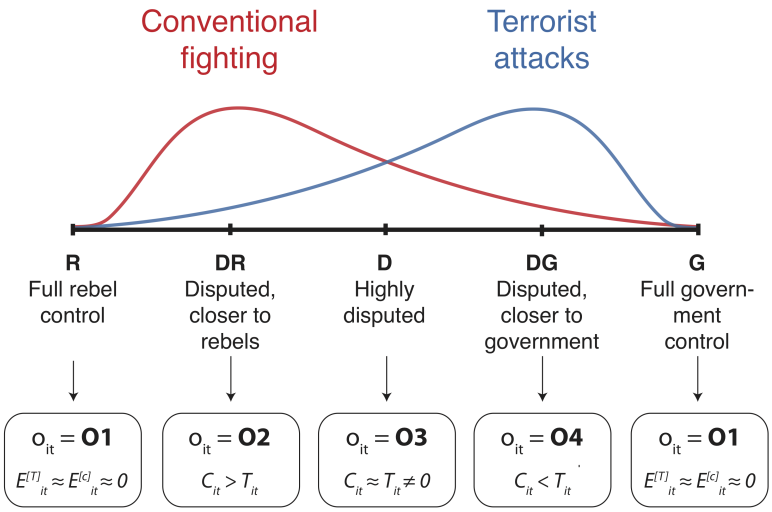


Figure 3 – Tactiques & contrôle territorial – Anders, 2020 [1]

4 L'algorithme de Viterbi

4.1 Le principe de l'algorithme

Le principal algorithme sur lequel nous avons travaillé au cours de ce projet informatique est l'algorithme de Viterbi, utilisé dans le cadre des chaînes de Markov cachées pour les problèmes qui relèvent du

« décodage ». Il est présenté dans le document hmm.py. Il s'agit, étant donné des observations (ici relatives au type de combats recensés en Irak et au Nigeria au cours d'une année), ainsi qu'une chaîne de Markov cachée, de déterminer la séquence d'états la plus probable (appelée « chemin ») ayant permis de générer les observations.

Une première solution à ce problème pourrait revenir à déterminer toutes les séquences d'état ayant pu générer la séquence d'observations, puis à calculer leurs probabilités mais cela représente un coût important. Une telle méthode aurait en effet un coût de $\mathcal{O}(N^T)$. C'est pourquoi l'algorithme de Viterbi y est préféré. Il construit ce qu'on appelle un treillis constitué de points donnant la probabilité que la chaîne de Markov cachée soit à l'état s selon une certaine séquence d'observations. On détermine, pour chaque sommet du treillis, la séquence la plus probable d'états cachés en fonction des données observées. Chaque cellule du treillis [s, k] représente la probabilité que la chaîne soit à l'état s après avoir parcouru les k premières observations et après être passée par la séquence d'états la plus probable de la chaîne de Markov. La valeur de chaque cellule du treillis est calculée en prenant récursivement le chemin le plus probable qui pourrait nous conduire à cette cellule. Nous représentons le chemin le plus probable en prenant le maximum sur toutes les séquences d'états précédentes possibles. L'algorithme de Viterbi retourne le treillis et que la séquence d'états la plus probable. Nous calculons cette séquence d'états en gardant une trace des observations qui ont conduit à chaque état, puis à la fin en retraçant le meilleur chemin depuis le début, à l'aide de la matrice appelée matrice de backtracing.

4.2 Implémentation

L'algorithme repose sur différentes étapes. Tout d'abord, il est nécessaire de définir les éléments caractéristiques des chaînes de Markov présentés dans la Table 1 qui vont être nécessaires pour faire tourner l'algorithme. Il s'agit donc tout d'abord de créer l'espace de probabilité dans lequel évolue la chaîne de Markov cachée avec les vecteurs donnant les états observables et les états cachés de la chaîne, ainsi que les probabilités initiales d'observer chaque état. Ces probabilités initiales sont différentes selon que l'on s'intéresse au Nigeria ou à l'Irak. C'est ce que la fonction `get_pi()` nous permet de mettre en évidence. Le module pandas est utilisé pour créer un tableau regroupant les valeurs et les probabilités des états cachés.

```
# creation de l'espace de probabilite
observable_states= ['O1', 'O2', 'O3', 'O4']
hidden_states = ['R', 'DR', 'D', 'DG', 'G']

# fonction permettant d'avoir les probabilites initiales du pays concerne
def get_pi(nom_pays) :
    if nom_pays == 'Iraq':
        pi = [0.2, 0.2, 0.2, 0.2, 0.2]
    elif nom_pays == 'Nigeria' :
        pi = [0.025, 0.025, 0.025, 0.025, 0.9]
    else:
        pi = [0.2, 0.2, 0.2, 0.2, 0.2]
    return pi
```

Pour le Nigéria, nous supposons un fort contrôle gouvernemental du pays en 2008. En revanche, pour l'Irak, nous supposons que le territoire est dans l'ensemble assez contesté en 2011.

Il s'agit ensuite de définir la matrice de transition et la matrice d'émission de la chaîne de Markov cachée en utilisant la fonction DataFrame du module pandas. Nous avons choisi ici de conserver les matrices de transition et d'émission employées par Thérèse Anders dans son article. La matrice de transition spécifie les probabilités de transition d'un état latent à un autre. La matrice d'émission précise comment la variation observée dans l'occurrence des tactiques terroristes et non terroristes est liée aux niveaux non observés de contrôle territorial. Chaque probabilité d'émission dans la matrice permet de savoir, étant donné que le véritable état caché au temps t est, par exemple, un contrôle rebelle complet R, la probabilité d'observer, par exemple, un signal de non-violence O_1 à partir des données . Anders dérive la matrice de transition des analyses des transitions entre les zones de contrôle territorial effectuées par Kavylas sur la guerre civile grecque en les adaptant à sa définition du contrôle territorial. Quant à la matrice d'émission, Anders la dérive de façon heuristique d'une analyse empirique de la situation qu'elle a pu faire.

```
t_df = pd.DataFrame(columns=hidden_states, index=hidden_states)
t_df.loc[hidden_states[0]] = [0.250, 0.500, 0.025, 0.200, 0.025]
t_df.loc[hidden_states[1]] = [0.250, 0.150, 0.075, 0.500, 0.025]
t_df.loc[hidden_states[2]] = [0.050, 0.025, 0.050, 0.850, 0.025]
t_df.loc[hidden_states[3]] = [0.025, 0.075, 0.150, 0.125, 0.625]
t_df.loc[hidden_states[4]] = [0.050, 0.075, 0.475, 0.025, 0.375]
t = t_df.to_numpy(dtype='float64')

e_df = pd.DataFrame(columns=observable_states, index=hidden_states)
e_df.loc[hidden_states[0]] = [0.600, 0.175, 0.175, 0.050]
e_df.loc[hidden_states[1]] = [0.050, 0.600, 0.175, 0.175]
e_df.loc[hidden_states[2]] = [0.050, 0.175, 0.600, 0.175]
e_df.loc[hidden_states[3]] = [0.050, 0.175, 0.175, 0.600]
e_df.loc[hidden_states[4]] = [0.600, 0.050, 0.175, 0.175]
e = e_df.to_numpy(dtype='float64')
```

	R	DR	D	DG	G
R	0.250	0.500	0.025	0.200	0.025
DR	0.250	0.150	0.075	0.500	0.025
D	0.050	0.025	0.050	0.850	0.025
DG	0.025	0.075	0.150	0.125	0.625
G	0.050	0.075	0.475	0.025	0.375

TABLE 5 – Matrice de transition

	O_1	O_2	O_3	O_4
R	0.600	0.175	0.175	0.050
DR	0.050	0.600	0.175	0.175
D	0.050	0.175	0.600	0.175
DG	0.050	0.175	0.175	0.600
G	0.600	0.050	0.175	0.175

TABLE 6 – Matrice d'émission

La fonction `Digraph()` du module graphviz nous permet encore une fois ici de visualiser par exemple la matrice de transition entre les différents états cachés sous la forme d'un graphe probabiliste (Figure 4).

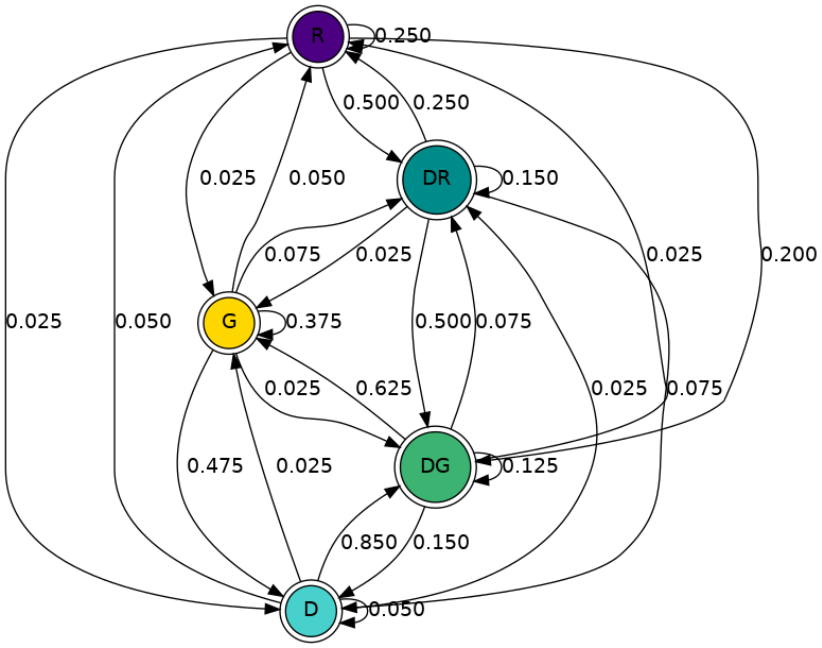


FIGURE 4 – Graphe probabiliste

Nous codons ensuite de façon numérique les observations et les états cachés

```
obs_map = {'O1':0, 'O2':1, 'O3':2, 'O4':3}
state_map = {0:'R', 1:'DR', 2:'D', 3:'DG', 4:'G'}
```

C'est seulement une fois ces étapes préalables effectuées que nous pouvons implémenter l'algorithme de Viterbi. Nous l'implémentons comme une fonction qui prend quatre paramètres, `(pi, t, o, obs)`,

`obs` étant la séquence d'observation calculée dans le programme `observation.py`, et qui retourne trois paramètres :

— la séquence d'états la plus probable ayant permis de générer les observations : path
— le treillis
— la matrice de backtracing qui permet d'effectuer de revenir en arrière sur les états cachés

Nous allons raisonner par récurrence sur des matrices. C'est pourquoi nous commençons la fonction par définir les paramètres que nous cherchons à déterminer comme des matrices vides dont la taille dépend du nombre d'état et du nombre d'observations.

```
N = np.shape(e)[0] #nombre d'etats
T = np.shape(obs)[0] #nombre d'observations
path = np.zeros(T)
treillis = np.zeros((N, T))
backtracing = np.zeros((N, T))
```

Puis nous implémentons l'algorithme selon la logique détaillée précédemment.

```
for s in range(N):
    # Initialisation de la construction du treillis
    for s in range(N):
        treillis[s, 0] = pi[s] * e[s, obs[0]]

    # Construction du treillis et de la matrice de backtracing
    for k in range(1, T):
        for s in range(N):
            treillis[s, k] = np.max(treillis[:, k-1] * t[:, s]) * e[s, obs[k]]
            backtracing[s, k] = np.argmax(treillis[:, k-1] * t[:, s])

    # Initialisation de la construction du chemin
    path[T-1] = np.argmax(treillis[:, T-1])

    #Backtracking
    for k in range(T-2, -1, -1):
        path[k] = backtracing[path[k+1], [k+1]]
```

Il existe une autre façon d'écrire l'algorithme de Viterbi faisant intervenir le logarithme. En effet, puisque toutes les valeurs de probabilité se situent dans l'intervalle $[0, 1]$, le produit de ces valeurs décroît de façon exponentielle avec le nombre n d'itérations. En conséquence, pour les séquences d'entrée d'observations importantes, les valeurs du treillis deviennent généralement extrêmement petites, ce qui peut poser des problèmes de précision lors des calculs. Une astuce, lorsqu'il s'agit de produits de valeurs de probabilité est alors de travailler dans le domaine log. On applique un logarithme à toutes les valeurs de probabilité et on remplace la multiplication par la sommation. Puisque le logarithme est une fonction strictement monotone, les relations de classement sont conservées dans le domaine log, ce qui permet de transférer des opérations telles que la maximisation ou la minimisation. Les résultats obtenus à la sortie des deux algorithmes sont bien les mêmes.

4.3 Tests unitaires

Le document `tests_unitaire.py` contient deux tests unitaires que nous avons réalisés sur l'algorithme de Viterbi. Ces tests sont bien entendus théoriques et ne reflètent aucune vraisemblance. Le premier consiste à transformer la matrice de transition en une matrice identité. L'algorithme de Viterbi doit nécessairement ressortir une séquence d'états identiques car la probabilité de passer d'un état caché à l'autre est nulle. Le second a consisté à transformer la matrice d'émission de façon à ce que la probabilité que l'état G soit réalisé en fonction de l'observation O_4 soit égale à 1. L'algorithme de Viterbi, pour une une séquence d'observation uniquement constituée d'attentats, doit ressortir une séquence d'états montrant uniquement un contrôle étatique.

4.4 Complexité de l'algorithme de Viterbi

L'algorithme de Viterbi est un algorithme issu de la programmation dynamique permettant de déterminer le chemin le plus probable associé à une séquence d'observations de façon particulièrement peu coûteuse. Sa complexité totale dépend des deux boucles `for` imbriquées : la première boucle est de longueur T, la seconde de longueur N et les opérations à l'intérieur de ces boucles ont une complexité en $\mathcal{O}$(N) (recherche d'un maximum) pour un coût total de $\mathcal{O}(\mathrm{N}^2\mathrm{T})$. Les autres opérations de l'algorithme ont un coût inférieur ($\mathcal{O}$(N)).

5 Calcul de la matrice d'exposition

Dans cette partie, nous présentons la manière dont nous calculons les matrices d'exposition du pays aux deux types de combats armés : le terrorisme et la guérilla.

5.1 Modélisation théorique

On subdivise l'espace en cellules hexagonales que nous traitons individuellement : on définit l'exposition d'une cellule i à la date t à l'ensemble des évènements de type terrorisme ou conventionnels comme une somme pondérée du nombre d'évènements ayant eu lieu à proximité de la cellule, et bien sûr avant la date t. La fréquence à laquelle nous effectuons ce calcul pour chaque cellule est le mois ; elle reste cependant ajustable dans le programme.

On construit ainsi une matrice E (pour exposure) de taille N × T, où N est le nombre de cellules et T est le nombre de mois considérés. Le scalaire E_{it} correspond à l'exposition de la cellule i au temps t. On pose :

$$\mathrm{E}_{it} = \sum_j w_{ij}^a w_{ij}^d \qquad (*)$$

où w_{ij}^a et w_{ij}^d sont respectivement les poids associés à l'âge de l'évènement (en mois) et à la distance entre l'évènement j et le centre de la cellule i.

Anders [1] suppose que l'effet d'une action de combat se dissipe de manière continue dans l'espace et dans le temps, et modélise alors la dissipation de l'impact d'un évènement avec une fonction logistique. En d'autres termes, on calcule des poids temporels w_{ij}^a (a pour *age*) et spatiaux w_{ij}^d de chaque évènement pour chaque cellule à l'aide d'une fonction de la forme :

$$w(x) = \frac{1}{1 + e^{-\kappa + \gamma x}}$$

où κ et γ sont choisis heuristiquement.

La formule de harvesine ci-dessous permet de mesurer la distance d entre deux points 1 et 2 à partir de leur latitude et longitude :

$$d = 2r \arcsin \sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}$$

où r est le rayon de la sphère ; φ_1, φ_2 sont les latitudes des points 1 et 2 (en radians) ; λ_1, λ_2 sont les longitudes des points 1 et 2 (en radians).

5.2 Implémentation pratique

Ces opérations sont effectuées par le programme `exposure.py`, dont nous en décrivons ici les grandes lignes.

Fonctions de calculs de base

Nous commençons notre programme par une définition des fonctions `harvesine()`, `wd()` et `wa()` permettant d'effectuer les calculs de distance et des poids spatiaux et temporels.

Quadrillage d'une aire

Pour subdiviser la surface d'un pays en hexagones, nous faisons appel au module H3, écrit par Uber. H3 met à disposition une fonction `polyfill()` qui, étant donnée une surface (représentée sous la forme d'un polygone shapely) et une résolution, renvoie l'ensemble des identifiants des cellules situées à l'intérieur de cette surface. Les frontières géographiques d'un pays sont chargées depuis un fichier geoJSON. On peut alors récupérer l'ensemble des identifiants des cellules situées à l'intérieure d'une zone géographique.

Calcul des distances entre cellules et évènements

Le calcul des matrices d'exposition se fait en deux temps :

1. On commence par calculer la matrice des distances du centre de chaque cellules aux évènements qui lui sont proches ;
2. On calcule ensuite l'exposition de chaque cellule i, à chaque date t, en calculant les poids w_{ij}^a et w_{ij}^d.

La fonction w^d tendant assez rapidement vers 0 au delà de quelques dizaines de kilomètres, on se contente pour chaque cellule de calculer la distance de son centre aux évènements qui sont suffisamment proches. On s'épargne ainsi un grand nombre de calculs inutiles. Ainsi, pour chaque cellule, la procédure est la suivante :

1. Déterminer l'ensemble des cellules voisines (et voisines des voisines) `ring` à l'aide de `h3.k_ring()` ;
2. Sélectionner les évènements situés à l'intérieur du `ring` ;
3. Calculer les distances de chaque évènement au centre de la cellule, puis le poids associé. Cette étape se fait sur des DataFrames pandas à l'aide d' `.apply()` .

```
def calc_distances(cells, events):
    distances = pd.DataFrame(index=events.index)
    distances['h3'] = events['h3'].copy()

    def calc_event_distance(origin, row):
        ...

    for cell in cells:
        ring = h3.k_ring(cell, 2)
        c_events = events.loc[(events['h3'].isin(ring))].copy()

        c_events['distance'] = c_events.apply(lambda row: calc_event_distance(cell, row), axis =
        ↪  1, result_type='reduce')
        c_events['wd'] = c_events.apply(lambda row: wd(row['distance']), axis = 1,
        ↪  result_type='reduce')

        distances[cell] = c_events['wd']

    return distances.drop('h3', axis=1)
```

Calcul de l'exposition

La procédure pour calculer l'exposition d'une cellule à une date donnée est essentiellement la même que pour le calcul des distances : on sélectionne les évènements proches et du type qui nous intéresse (terrorisme ou conventionnel), on calcule le poids associé à l'âge de l'évènement, et, connaissant les poids w^d, on en déduit l'exposition selon la formule ($\star$).

```python
def cell_exposure_at_t(events, attack_type, origin, date, distances):
    ring = h3.k_ring(origin, 2)
    c_events = events.loc[(events['h3'].isin(ring)) & (events['type'] == attack_type)].copy()

    def calc_event_age(row):
        return date -row['date_start']

    def calc_wa(row):
        # Calcule le poids de l'évènement en fonction de son âge
        ...

    c_events['age'] = c_events.apply(calc_event_age, axis = 1, result_type='reduce')
    c_events['wa'] = c_events.apply(calc_wa, axis = 1, result_type='reduce')

    wd_df = distances[origin].loc[(events['h3'].isin(ring)) & (events['type'] == attack_type)]

    return np.dot(wd_df, c_events['wa'])
```

De même, cette étape se fait sur des DataFrames pandas à l'aide d' `.apply()` .

On peut finalement écrire, à l'aide de toutes les fonctions précédentes, une fonction `get_exposure()` pour calculer la matrice complète. L'idée est de construire un DataFrame avec en élément, pour chaque colonne `date` et ligne `h3` , le couple (`date` , `h3`), puis d'appliquer la fonction `cell_exposure_at_t()` sur chacun de ces éléments.

```python
def get_exposure(events, attack_type, p: prm.Parameters, freq):
    cells = []
    for country in p.countries:
        cells += list(json_to_h3(country, p.h3_level))

    date_range = pd.date_range(p.startdate, p.enddate, freq=freq)

    exposure = pd.DataFrame(data = [[(h3, date) for date in date_range] for h3 in cells],
                            index = cells,
                            columns = date_range)

    distances = calc_distances(cells, events)

    def calc_exposure(c):
        origin, date = c[0], c[1]
        return cell_exposure_at_t(events, attack_type, origin, date, distances)

    return exposure.progress_applymap(calc_exposure)
```

5.3 Analyse des performances

Nous manipulons de grands tableaux de données, sur lesquels nous appliquons des fonctions pour chaque entrée du tableau. Il nous est apparu nécessaire d'estimer les temps d'exécution des différentes étapes dans le calcul de l'exposition afin de déterminer les sources de lenteurs. Nous présentons ici une courte analyse de chacune des étapes décrites dans la partie précédente.

Nous prenons pour nos mesure le cas de la Syrie, entre le 1er janvier et le 31 décembre 2015. A titre indicatif, les mesures ont été faites sur un AMD Ryzen 5 3600 (3.6 GHz).

5.3.1 Quadrillage d'une aire

Les fonctions ne sont pas complètement triviales. Par exemple, on peut s'attendre à ce que la détermination des cellules à l'intérieure d'une aire soit assez longue. En pratique, le module H3 est bien implémenté en Python, si bien que cette étape s'exécute avec célérité.

```
%time _ = country_grid("Syria", 6)
CPU times: user 115 ms, sys: 3.91 ms, total: 119 ms
Wall time: 108 ms
```

5.3.2 Calcul des distances entre cellules et évènements

Le calcul des distances, bien qu'optimisé dans la mesure où l'on ne regarde pour chaque cellule que les évènements suffisamment proche, prend une bonne quinzaine de secondes.

```
%time distances = calc_distances(cells, events)

CPU times: user 14.2 s, sys: 727 ms, total: 15 s
Wall time: 15 s
```

On peut faire quelques commentaires :

— Pour déterminer `c_events`, deux parcours complets d'`events` sont nécessaires : un premier pour déterminer l'ensemble des évènements dans le `ring`, puis un second pour sélectionner ces évènements.
— Peu d'informations sont données sur le type d'objet renvoyés par les fonctions utilisées dans les `.apply()`.
— Les manipulations sur les tableaux se font sur des données de types variés : `str` pour `cell`, `float` pour les latitudes et longitudes.

5.3.3 Calcul de l'exposition

En combien de temps se fait le calcul de l'exposition d'une cellule à une date donnée ?

```
%time _ = cell_exposure_at_t(events, attack_type, origin, date, distances)

CPU times: user 11.2 ms, sys: 0 ns, total: 11.2 ms
Wall time: 10.4 ms
```

On mesure une petite dizaine de millisecondes. Cela peut paraître peu, mais combien de fois ce calcul doit-il être fait ? Pour l'exemple de la Syrie, nous avons 4712 cellules que nous analysons pendant 12 mois. Cela fait un total de $4712 \times 12 = 56544$ cellules, et donc un temps d'exécution de 565,44 secondes soit environ 9 minutes et 30 secondes. Pour une analyse sur une période de 10 ans, cela nous prendrais donc environ une centaine de minutes pour calculer la matrice d'exposition à un type de combat, et donc le double en temps puisqu'on considère terrorisme et guérilla. Un tel temps d'attente n'est clairement pas raisonnable.

Nous nous sommes donc interrogés sur la manière dont nous pourrions accélérer ce processus.

6 Optimisation des performances à l'aide de Numba

Python est connu pour être à la fois un langage très accessible, et aussi très lent – les deux allant de pair. Rappelons qu'une source principale de lenteur vient du fait que le langage est très peu « typé ». Par exemple, les listes peuvent être des listes d'éléments de types radicalement différents les uns des autres : `[3, "Bonjour", lambda x: x+1]` est une liste valide en Python ; et les fonctions peuvent agir sur des éléments de types différents (un coup un `int`, un autre coup un `float`), et renvoyer des éléments eux-aussi de types différents.

Ces propriétés de Python, qui font aussi sa force, sont sources de lenteur car, d'une part, l'intérpréteur doit souvent deviner le type d'objet auquel il a affaire avant d'exécuter une fonction, et, d'autre part, la représentation en mémoire des objets Python doit prendre en compte la possibilité de types multiples,

ce qui réduit leur efficacité. Pire encore, le langage étant interprété et non compilé, ce travail doit être effectué à chaque fois qu'une fonction est appelée, et non une fois pour toutes au début de l'exécution du programme.

Pour accélérer notre code, une solution est donc de court-circuiter ce processus en travaillant avec des tableaux à types déterminés, et en compilant nos fonctions à la première exécution pour que les appels suivant se fasse plus rapidement. Nous faisons pour cela appel aux bibliothèques NumPy et Numba.

6.1 « Numba makes Python code fast »

Voici la présentation du module, telle qu'elle est écrite sur le site internet : « Numba translates Python functions to optimized machine code at runtime using the industry-standard LLVM compiler library. Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN ».

La traduction se fait simplement par l'ajout du décorateur `@njit`. En général, le premier appel d'une fonction « jittée »prend plus de temps que sa version Python (c.-à-d. sans `@njit`) car il faut compter le temps de la compilation de la fonction. Une fois celle-ci réalisée, les appels suivant sont *significativement* plus rapides : on peut diviser le temps de calcul par un facteur de l'ordre de la centaine. Cette fonctionnalité est particulièrement adaptée à notre programme, qui fait massivement appel à la fonction `calc_cell_exposure_at_t()`.

Pour que la compilation par Numba se fasse efficacement, il est utile (et parfois nécessaire) d'indiquer explicitement quels types de données nous sommes en train de manipuler. Numba est ainsi conçu pour fonctionner en symbiose avec NumPy : Numba exploite en effet pleinement la représentation machine des `array` de NumPy qui, dès lors que le type de leurs éléments est bien défini, permet des opérations rapides sur ceci.

En pratique, pour assurer une représentation en mémoire efficace, on décide de ne plus utiliser de `pd.DataFrame` pour les remplacer uniquement par des `np.array`, et d'éviter au maximum d'utiliser des types non canoniques comme les `pd.Timestamp` représentant les dates pour se limiter aux `int`, `float` et parfois `str`. Par conséquent, on ne pourra plus utiliser les méthodes `.loc[]` des DataFrames pour trouver, filtrer et extraire les éléments d'un tableau. Il est alors nécessaire de reprendre complètement le code d'`exposure.py` : c'est ce qui est fait dans le nouveau fichier `exposure_numba.py`.

Afin que le processus de compilation par Numba se déroule sans encombre, on adopte les principes de programmation suivants :

- *Toutes les listes Python sont systématiquement converties en* `np.array` *avec un type clair* (par exemple `np.float64`, qui sont les flottants de NumPy sur 64 bits) ;
- *Pas de réallocation mémoire* : au lieu d'initialiser un tableau vide et d'ajouter au fur et à mesure des éléments, on préférera l'initialiser avec une taille fixe, en précisant bien sûr son type (par exemple un tableau rempli de `0.`, ou rempli de `True`) ;
- *Conversion de toutes les dates au format epoch*, c.-à.-d. en nombre de secondes depuis le 1er janvier 1970. Cela nous permet de manipuler des `int` et non des `pandas.Timestamp` ;
- *Pas de fonctions exotiques au sein des fonction jittées* : par exemple, on n'utilise pas les fonctions du module H3 pour calculer l'exposition d'une cellule.

6.2 En pratique

Redéfinition de variables

La première étape est d'ajuster la manière dont nous déclarions les variables dans la fonction `get_exposure()`, de sorte qu'à n'utiliser que des tableaux NumPy. On définit et calcule ainsi les nouvelles variables suivantes :

- Un tableau `cells_coord` à deux colonnes (latitude, longitude) contenant les coordonnées du centre des cellules ;
- La même chose pour les évènements avec `events_coord` ;
- Un tableau `is_in_ring` tel que si l'évènement j a lieu dans le `h3.k_ring()` de la cellule i, alors `is_in_ring[i][j]` vaut `True` : cela évite de faire appel à H3 au sein des fonctions calculant l'ex-

position d'une cellule ;
— Deux tableaux `date_range` et `events_dates` contenant des dates au format epoch (et donc des `int`).

```
cells = []
for country in p.countries:
    cells += list(json_to_h3(country, p.h3_level))

cells_coord = np.array([[h3.h3_to_geo(c)[0], h3.h3_to_geo(c)[1]] for c in cells],
↪  dtype='float64')
events_of_type = events[events["type"] == attack_type]
events_coord = events_of_type[["latitude", "longitude"]].to_numpy(dtype='float64')

events_h3 = events_of_type["h3"].to_numpy()
rings = np.array([list(h3.k_ring(str(cell), 2)) for cell in cells], dtype='object')
is_in_ring = np.array([np.isin(events_h3, ring) for ring in rings])

date_range = pd.date_range(p.startdate, p.enddate, freq=freq)
date_range = ((date_range - pd.Timestamp("1970-01-01")) // pd.Timedelta("1s")).to_numpy()

events_dates = ((events["date_start"] - pd.Timestamp("1970-01-01")) //
↪  pd.Timedelta("1s")).to_numpy()
```

Nouveau calcul des distances

La nouvelle fonction de calcul des distances est très simple :

```
@njit
def calc_distances(cells_coord, events_coord, is_in_ring):
    res = np.zeros((len(cells_coord), len(events_coord)), dtype='float64')
    for i in range(len(cells_coord)):
        for j in range(len(events_coord)):
            if is_in_ring[i,j]: # if events_coord[j] is in ring(cells_coord[i])
                lat1, lng1 = cells_coord[i][0], cells_coord[i][1]
                lat2, lng2 = events_coord[j][0], events_coord[j][1]
                res[i,j] = harvesine(lng1, lat1, lng2, lat2)

    return res
```

Cette nouvelle représentation des données nous permet déjà de gagner un temps non négligeable dans le calcul du tableau de distances :

```
%time distances = calc_distances(cells_coord, events_coord, is_in_ring)

CPU times: user 169 ms, sys: 7.82 ms, total: 177 ms
Wall time: 176 ms
```

On est ainsi passé d'un temps de calcul d'une quinzaine de secondes avec l'implémentation précédente, à quelques centaines de microsecondes !

Nouveau calcul de l'exposition

Le code est également simple. On notera l'initialisation des `wa_array` et `wd_array` en tableaux vides, selon le principe indiqué plus haut.

```
def cell_exposure_at_t(events_dates, cell_index, date, is_in_ring, distances):
    wa_array = np.zeros(n_events, dtype='float64')
    wd_array = np.zeros(n_events, dtype='float64')
```

```
    for j in range(n_events):
        if is_in_ring[cell_index, j]:
            age = date - events_dates[j]
            wa_array[j] = wa(age)
            wd_array[j] = wd(distances[cell_index, j])

    return np.dot(wd_array, wa_array)
```

```
%time _ = cell_exposure_at_t(events_dates, cell_index, date, is_in_ring, distances)

CPU times: user 51 us, sys: 0 ns, total: 51 us
Wall time: 45.3 us
```

Le nouveau calcul de l'exposition se fait désormais en une dizaine de microsecondes : c'est 1000 fois plus rapide que l'implémentation précédente.

Il faut enfin faire le lien entre les nouvelles fonctions et le design de code d'`exposure.py`. Ici, `get_exposure_raw()` produit une matrice d'exposition de la forme d'un tableau NumPy, et `get_exposure()` est en fait un wrapper autour de cette première fonction. Il renvoie à la fin un DataFrame, le format utilisé dans les autres programmes du projet.

```
def get_exposure_raw(events_dates, is_in_ring, distances, date_range):
    n_cells, n_events = distances.shape
    n_dates = len(date_range)
    exposure = np.zeros((n_cells, n_dates), dtype='float64')

    for i in range(n_cells):
        for d in range(n_dates):
            exposure[i][d] = cell_exposure_at_t(events_dates, i, date_range[d], is_in_ring,
            ↪  distances)

    return exposure

def get_exposure(events, attack_type, p, freq):
    # ...
    # Les nouvelles variables mentionnees precedemment sont definies ici
    # ...
    g = get_exposure_raw(events_dates, is_in_ring, distances, date_range)

    return pd.DataFrame(g, index=cells, columns=pd.to_datetime(date_range, unit="s"))
```

```
%time exposure = get_exposure(events, attack_type, p, freq)

CPU times: user 2.33 s, sys: 3.57 ms, total: 2.34 s
Wall time: 2.35 s
```

Le nouveau temps de calcul de la matrice d'exposition est de l'ordre de 3 secondes, ce qui est environ *250 fois plus rapide* que l'implémentation précédente.

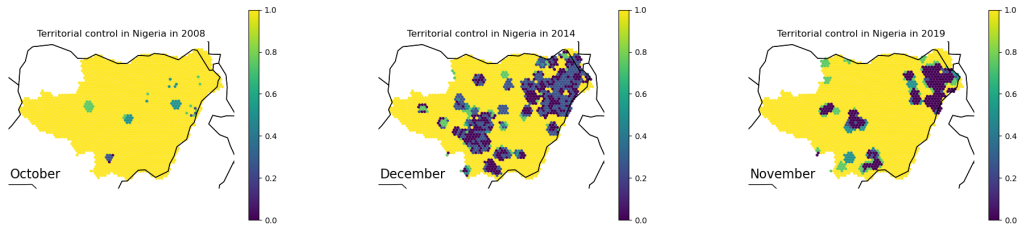
6.3 En conclusion

La nouvelle implémentation du calcul des matrices d'exposition à l'aide de Numba nous permet d'obtenir ces matrices dans des délais très raisonnables, ce qui est non-négligeable dans la mesure où nous avons régulièrement du faire des essais et ajustements au cours de notre travail. Le prix à payer est un code dont la syntaxe s'éloigne du style Python canonique pour se rapprocher de ce qu'on peut trouver en C++ par exemple ; c'est parfaitement acceptable vu les gains obtenus.

7 Description des résultats obtenus

À partir du calcul de la matrice d'exposition et des séquences d'états produites par l'algorithme de Viterbi, nous produisons des cartes mensuelles permettant d'observer le contrôle territorial de l'Irak entre 2011 et 2019 et du Nigeria entre 2008 et 2019. Le contrôle territorial est défini par une échelle de couleur allant de 0 à 1, 0 représentant un contrôle territorial entièrement rebelle (R), 1 représentant un contrôle territorial entièrement étatique (G).

Nous présentons en Figure 5 trois cartes de contrôle territorial au Nigéria produites par notre programme, à trois périodes marquantes de l'insurrection.

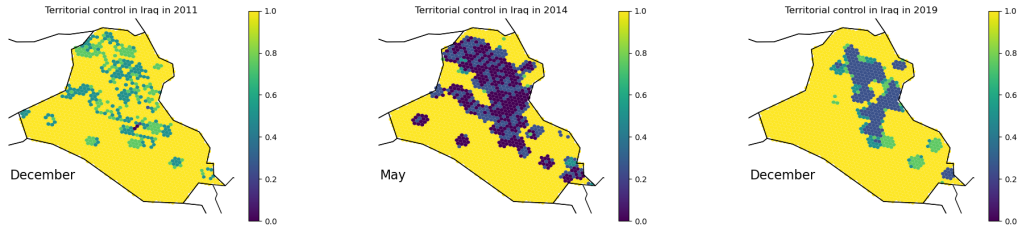


(a) (b) (c)

FIGURE 5 – Contrôle territorial au Nigéria en (a) octobre 2008 (b) décembre 2014 et (c) novembre 2019

Les premières actions de Boko Haram témoignant d'une prise de contrôle du territoire semblent dater de la fin de l'année 2008 et du début de l'année 2009 comme en témoigne la présence de cellules bleues et mauves dès 2008 sur les cartes. Ce contrôle reste timide jusqu'à la fin de l'année 2010 et se localise sur des poches de territoires très localisées (comme les États de Borno à l'Est et de Yobe au Nord). Le contrôle de Boko Haram (et avec, ensuite, celui de l'ISWAP) sur le territoire va s'accentuer à partir de 2011 et jusqu'en 2014 et 2015 où le groupuscule contrôle une grande partie du territoire concentrée dans le Nord Est du pays. On observe également à partir de 2017 une présence rebelle importante dans le Sud du pays. A partir de 2019, l'influence de Boko Haram commence à décroître sur l'intégralité du territoire national.

De même, pour l'Irak, trois cartes produites à trois moments clés de la lutte contre l'État Islamique sont données en Figure 6.



(a) (b) (c)

FIGURE 6 – Contrôle territorial en Irak en (a) décembre 2011 (b) mai 2014 et (c) décembre 2019

Jusqu'en août 2011, en Irak, aucun territoire du pays est contrôlé par les rebelles (et donc par l'Etat islamique) comme en témoigne l'absence de cellules mauves et bleues foncées qui en sont caractéristiques. Toutefois, dès 2011, ces derniers semblent déjà actifs dans la mesure où le territoire national semble bien plus contesté que le territoire Nigérien en 2009. Les premières cellules rebelles apparaissent en septembre et augmentent de façon progressive en 2012 puis se stabilisent. C'est véritablement à partir de juin 2013 que les acteurs rebelles gagnent du territoire de façon très importante au détriment de l'Etat. A partir de 2014, l'Etat islamique semble contrôler de façon assez constante près d'un tiers du territoire irakien, principalement dans le Nord-Est du pays à la frontière avec la Syrie, et notamment les régions de Ninewa et Anbar, même si le contrôle s'étend également par poches, et selon les périodes, dans le Sud (probablement autour des villes qui sont le lieu de la plupart des évènements violents, montrant l'un des biais du modèle). C'est seulement depuis 2019 que le contrôle de l'Etat Islamique semble décroître, bien que les interventions militaires armées visant à libérer le pays aient débuté bien avant. Il faut en effet noter que sur nos cartes,

l'influence des groupes djihadistes reste très importante entre autres au centre du pays, et ce bien après la reprise de certains territoires par les forces armées kurdes. On peut se questionner sur ces résultats, et se demander s' ils sont la conséquence des répercussions armées sur le territoire. Cela pourrait représenter une des limites du modèle implémenté.

Dans l'ensemble, cette étude de carte semble refléter de façon plutôt efficace les actualités qui ont régi les périodes observées pour les deux pays. Les résultats que l'on obtient pour le Nigeria recoupent par ailleurs assez bien les résultats obtenus par Thérèse Anders dans sa thèse. Dans les deux cas, on visualise de façon claire les périodes de montée en puissance des groupuscules contestataires, l'apogée de leur influence, et puis leur déclin. Le contrôle territorial de ces derniers est dans les deux cas très localisé (Nord pour l'Irak, Nord Ouest et Sud Est pour le Niger), et ce de façon assez constante. Il faut cependant souligner que dans les deux cas les zones signalées comme contrôlées par les rebelles ne sont pas toujours le fait de Boko Haram pour le Nigéria et de l'État islamique pour l'Irak. Par exemple pour le Nigéria, l'Ipob, le mouvement indépendantiste biafrai et sa branche armée le Réseau de sécurité de l'Est - l'ESN - ont souvent opéré sur la période étudiée dans le Sud-Est du pays par exemple dans les États d'Abia, Akwa Ibom ou Imo. Ce qui pourrait expliquer la force rebelle persistante dans le Sud du pays sur les cartes. Il est intéressant de noter que pour les deux pays, les contrôles territoriaux varient énormément d'un mois à l'autre. Cela peut être interprété comme une marque spécifique des conflits asymétriques pour lesquels les contrôles sont très fluctuants, et donc pour lesquels il est toujours difficile de délimiter les zones de contrôle par frontières précises. C'est l'un des avantages très net de cette modélisation de permettre de bien saisir toutes les fluctuations temporelles qui peuvent exister dans les zones de conflits asymétriques, et qui sont difficiles à saisir par de simples observations.

8 Prolongements

Ce projet ne relève pas d'une expérience aléatoire ou encore d'une optimisation, où le but de l'implémentation du projet est de parvenir à trouver un résultat inattendu ou encore une solution au problème. Ici, le résultat étant d'une certaine façon déjà connu, l'enjeu réside dans la capacité à modéliser de façon efficace un événement. Ce projet nous a donc invités à trouver des façons de rendre notre travail le plus représentatif possible de la situation en Irak et au Nigeria.

Ce travail va toutefois au-delà d'une simple modélisation, sinon le projet aurait assez peu d'intérêt. La méthode proposée par Thérèse Anders apparaît comme un moyen très intéressant de prendre en compte les évolutions du contrôle territorial d'une zone contestée dans la mesure où il repose simplement sur un recensement des conflits dans la zone concernée. D'autant plus que les estimations du contrôle d'un territoire contesté sont très approximatives car il est difficile de se rendre sur place et de déterminer entre les différents acteurs en jeu lequel est le plus influent. De nombreuses améliorations de l'algorithme sont possibles, que nous n'avons pas forcément pu proposer dans ce projet par manque de temps ou encore de données publiques. Il serait possible par exemple d'exploiter les données relatives au gain territorial des acteurs gouvernementaux et non-gouvernementaux lors des actions menées lors des conflits territoriaux, recensées par la base de données ACLED (*Armed Conflict Location and Event Data Project*) pour effectuer une comparaison avec les résultats obtenus dans le projet. Le nombre de victimes des combats n'est, dans ce modèle, pas pris en compte ; or il est un bon indicateur de l'importance relative de chaque évènement. Il serait intéressant de chercher à l'intégrer dans notre modèle. Il serait également intéressant d'utiliser ce modèle pour étudier les frontières contestées entre deux entités nationales comme Israël et la Palestine.

Cette modélisation rencontre certaines limites. Tout d'abord, le fait de modéliser le contrôle territorial autour d'un prisme bipolaire état/groupuscule rebelle peut très bien s'appliquer à de nombreux conflits asymétriques comme celui faisant rage entre Boko Haram et le Nigeria, ou encore entre le FARC et le gouvernement colombien mais pose problème dès lors que l'état fait face à plusieurs groupuscules rebelles comme cela peut être le cas d'une certaine façon davantage en Irak, mais aussi sur de nombreux autres territoires comme l'Afghanistan. Notamment, la modélisation en l'état ne peut s'appliquer à la Syrie, où se déroule la plus importante guerre civile de ce début de XXI^{e} siècle : on y retrouve en effet un nombre important d'acteurs combattants les uns contre les autres avec des méthodes variées, dépassant ainsi la bipartition « gouvernement contre insurgés ». Le modèle reste également approximatif pour ce qui est de déterminer nettement l'état d'un territoire. Les matrices d'émission et de transition sont tenues pour données, alors qu'elles pourraient être adaptées à chaque situation, et le fait d'attribuer un type d'évènement particulier (terrorisme, guerre conventionnelle) à un état de contrôle reste approximatif. Il faut aussi prendre en compte le fait que les évènements terroristes surviennent la plupart du temps davantage dans les villes que dans les endroits enclavés du territoire, ce qui biaise d'une certaine façon les résultats.

Enfin, le modèle reste intrinsèquement lié à une observation d'événements violents dans le cadre d'un conflit armé et ne prend pas en compte des changements de contrôle territorial qui pourraient être liés à des négociations et à des événements non violents, bien que ces derniers soient rares dans les situations qu'on étudie.

Dans l'ensemble, il n'existe pas de vérité fondamentale sur le contrôle territorial pour la plupart des conflits, en particulier pour les guerres civiles asymétriques. Cela réduit la fiabilité des estimations, car les paramètres du modèle ne peuvent pas être tirés des données déterminées avec exactitude, même si on observe une certaine constance dans le temps sur la logique de fonctionnement du contrôle territorial des zones de conflit asymétrique, comme en témoignent les matrices d'émission et de transition. Cependant, cela ouvre la voie à des simulations d'évolution des conflits sur le long terme. À titre d'exemple, le choix des probabilités initiales permet de faire évoluer le modèle d'une façon très différente. On peut faire débuter le modèle avec un contrôle du territoire de la part du gouvernement important, comme nous l'avons fait pour le Nigeria, mais on pourrait aussi dans d'autres contextes de conflit, faire fonctionner le modèle avec un contrôle préalable total des rebelles afin de produire des estimations théoriques et de contribuer à faire évoluer les recherches empiriques sur le contrôle territorial en temps de guerre civile.

Conclusion

Nous avons pris beaucoup de plaisir à travailler sur ce projet, qui nous a permis de lier une thématique géopolitique qui nous intéresse fortement à la *data science* qui est au cœur de notre formation à l'ENSAE. Ce projet ne s'est pas fait sans difficultés, notamment parce que la littérature et les données à ce sujet ne sont pas particulièrement développées ; mais cela nous a permis de prendre des initiatives en vue de construire un algorithme aussi efficace que possible.

Ce projet, croisant l'expérience de deux personnes issues de filières préparatoires profondément différentes (B/L et MP), nous a aussi beaucoup appris d'un point de vue personnel. Il nous a invités à croiser nos connaissances et nos méthodes de travail et à mettre à profit nos atouts au service de l'autre afin que chacun apprenne beaucoup du projet.

Dans l'ensemble, ce travail nous a donné un aperçu de ce qui pouvait nous attendre plus tard d'un point de vue professionnel, dans la mesure où nous nous sommes intéressés au domaine de la géopolitique, dans lequel nous aspirons tous les deux à travailler. La *data science* est aujourd'hui un élément essentiel de la recherche dans les relations internationales et de nombreux organismes, français (IFRI, IHEDN…) ou encore internationaux (Impact Research, Peace Research Institute Oslo…) recherchent de plus en plus des profils de *data scientists*. Nous espérons que ce projet que nous avons mené nous sera un vrai atout dans notre avenir professionnel.

Références

[1] T. Anders, "Territorial control in civil wars : Theory and measurement using machine learning," *Journal of Peace Research*, vol. 57, no. 6, pp. 701–714, 2020.

[2] S. M. Polo and K. S. Gleditsch, "Twisting arms and sending messages : Terrorist tactics in civil war," *Journal of Peace Research*, vol. 53, no. 6, pp. 815–829, 2016.

[3] D. Jurafsky and J. H. Martin, *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.* Pearson, 2021.

[4] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

Appendices

Exécution du code

L'ensemble des programmes sont à retrouver dans le dossier `src/`.

Prérequis

Les modules Python H3 et GeoPandas sont requis pour exécuter les différents programmes. Ils peuvent être installé via Anaconda :

```
conda install -c conda-forge h3-py geopandas
```

De plus, la version accélérée du programme de calcul des matrices d'expositions s'appuie sur Numba :

```
conda install -c conda-forge numba
```

Quickstart

```
$ cd src/
$ python wrapper.py

Choose a country : [I]raq | [N]igeria (default is Nigeria) > i
Enter startdate (YYYY-MM-DD, default is 2008-01-01) > 2011-01-01
Enter enddate (YYYY-MM-DD, default is 2019-12-31) >
Enter H3 level (0-15, default is 5) >
Enter frequency (default is M) >
Use numba ? Y/n >

(Chargement des données)
(Calculs de la matrice d'exposition, puis des probabilités)

The median of C is : 1.0
The median of T is : 0.26634408573507873
Enter a value for m (default is 0.15) > 0.3
Enter a value for xs (default is 0.01) >

(Calcul des états)
(Création des cartes)
```

Résumé des composantes

- `precleaning.py` : chargement des bases de données, premiers filtrages
- `exposure.py`, `exposure_numba.py` : calcul des matrices d'exposition
- `observation.py` : calcul des séquences d'observations
- `hmm.py` : estimation des séquences de contrôle territorial
- `figures` : production de graphiques

Si chaque programme peut être exécuté individuellement, le fichier `wrapper.py` permet d'éxécuter l'ensemble des programmes précédents séquentiellement.

Dans un terminal, depuis le dossier `src/` :

```
python wrapper.py
```

Présentation des composantes

Préparation des données (`precleaning.py`)

Le programme charge les deux bases de données et en extrait les variables nécessaires, en plus d'appliquer quelques opérations de bases (filtrage, calculs de dates).

Penser à s'assurer que les chemins des bases de données dans `precleaning.py` sont corrects. Par défault, ils sont de la forme `../datasets/*.csv`. Les bases de données sont à renommer en `ged.csv` et `gtd.csv`.

Calcul de l'exposition aux conflits (`exposure.py`)

Penser à s'assurer que les chemins des GeoJSON dans `exposure.py` sont corrects. Par défault, ils sont de la forme `../GeoJSONs/Country.geo.json`.

Ce fichier se charge du calcul d'exposition aux conflits. Les deux matrices obtenues après calcul (une pour l'exposition aux actes terroristes, une autre aux combats conventionnels) sont enregistrées dans le dossier `../exposures/`.

Le calcul est long : environ 6 minutes pour l'exposition sur un an à intervalles d'un mois.

Calcul de l'exposition aux conflits, version accélérée (`exposure_numba.py`)

Ce programme s'appuie sur Numba pour calculer rapidement les matrices d'exposition.

Le gain de temps est significatif : compter une petite dizaine de secondes pour l'exposition sur un an à intervalles d'un mois.

Détermination d'une séquence d'observation (`observation.py`)

A partir des matrices d'exposition générées par `exposure.py`, ce programme génère une séquence d'observation pour chaque cellule géographique, sur la période donnée.

Les résultats sont enregistrés sous la forme d'un tableau `.csv` dans le dossier `../observations/`.

Détermination d'une séquence de contrôle territorial (`hmm.py`)

Ce programme détermine une séquence sous-jacente de contrôle territorial, à partir d'une séquence d'observations et à l'aide d'un modèle de Markov (algorithme de Viterbi).

Les résultats sont enregistrés sous la forme d'un tableau `.csv` dans le dossier `../controls/`.

Production de graphiques (`figures.py`)

Ce programme produit un ensemble de cartes à partir des séquences de contrôle calculées par `hmm.py`.

Les résultats sont enregistrés sous la forme de cartes `.png` dans le dossier `../figures/`.

Des ajustements du code sont à prévoir au cas par cas selon les pays si l'on souhaite obtenir des cartes propres.