

Projet ”analyse tactique de trajectoires”

Étude de la distance entre deux segments

Réalisé par:

Collenot Antoine
Danvy Alix
Lecoindre Kenzo
Papa Samba Khary Touré

Projet scientifique, réalisé dans le cadre de l'UE projet, visant à réaliser une analyse de trajectoires (ici appliqué sur des données du jeu Dota 2). Pour y arriver, nous avons procédé avec les méthodes suivantes: compression s'appuyant sur le principe MDL, discréétisation par clustering (K-means, K-medoids ou Propagation d'affinités) et extraction séquentielle des motifs fréquents.

Licence 3 d'informatique 2023

Professeurs référents :

Mr.Rioult François
Mr.Paniol Marc



Contents

1	Introduction	4
2	Organisation du projet	5
2.1	Pipeline	5
3	Réalisation du pipeline	7
3.1	Compression de trajectoire	7
3.1.1	Propriétés désirables pour la compression de trajectoire	7
3.1.2	Fonction de distance	8
3.1.3	Principe MDL formalisé	8
3.1.4	Méthode de compression rapide en temps linéaire	8
3.2	Clustering	9
3.2.1	K-Means et K-Medoids	10
3.2.2	Propagation d'affinité	10
3.3	Extraction de connaissance dans les données	12
3.3.1	Extraction séquentielle	12
4	Expérimentations et Discussion	14
4.1	Compression	14
4.2	Clustering	15
4.2.1	Paramétrisation de la distance entre segments	15
4.2.2	Paramétrisation du nombre de clusters	19
4.3	Extraction séquentielle	21
4.3.1	Résultats	22
4.3.2	Discussion	23
5	Conclusion	24
5.1	Résultats	24
5.2	Améliorations	25
A	Compression MDL	26
A.1	Fonction de distance	26

A.2	Compression en temps linéaire	27
B	Propagation d'affinité	28
B.1	Implémentation	28
B.2	Méthode de calcul de la similarité	29
B.3	Méthode de calcul de la responsabilité	29
B.4	Méthode de calcul de la disponibilité	31

Chapter 1

Introduction

Depuis de nombreuses années, l'analyse de données est devenue un domaine clé de notre évolution. En effet, les sciences des données permettent de traiter de plus en plus aisément de grandes quantités de données et d'en extraire des connaissances pertinentes.

Dans notre projet, nous nous concentrerons sur un domaine spécifique de la science des données : l'analyse de trajectoires. Tout comme la génétique a utilisé la drosophile comme modèle pour comprendre les principes fondamentaux de l'hérédité, nous utiliserons les jeux vidéo pour explorer les principes sous-jacents de l'analyse de trajectoires, ici, il s'agira du jeu Dota 2.

Les jeux vidéo offrent un environnement idéal pour l'analyse de trajectoires, car ils fournissent des données riches et précises sur les mouvements des joueurs, leurs interactions avec l'environnement et les stratégies de jeu. En utilisant ces données, nous pouvons explorer les motifs de jeu, identifier les comportements des joueurs et améliorer à terme les performances tactiques.

L'identification des motifs de jeu requiert de discréteriser les trajectoires afin d'obtenir pour chaque joueur des suites de symboles plutôt que des coordonnées. Ce passage au symbolique permet de comparer les comportements collectifs grâce à des méthodes de découverte de motifs fréquents. Dans cette étude, nous utilisons donc des méthodes de compression, de clustering et d'extraction séquentielle pour analyser les trajectoires de Dota2.

Nos objectifs sont :

1. Illustrer les différences entre les deux algorithmes de clustering que nous utiliserons, à savoir K-means et Affinity Propagation.
2. Discuter de l'intérêt des régularités découvertes dans nos données.

La problématique est de réussir à mettre en place cette chaîne de traitements successifs des données et de paramétrier chaque étape pour obtenir des résultats pertinents.

Dans cette optique, nous allons donc commencer par expliquer l'organisation de ces traitements, puis nous développerons sur le fonctionnement de chacune de ces étapes, pour finalement regarder et discuter les résultats obtenus à chaque étape.

Chapter 2

Organisation du projet

Comme évoqué, l'objectif du projet de pouvoir extraire des connaissances à partir de nos données séquentielles. Pour y arriver, nous devons au préalable traiter successivement nos données pour les rendre exploitables, si possible en un temps raisonnable.

Les trajectoires disponibles sont très précises et représentent des suites de segments de déplacement à l'échelle de la seconde. Pour les rendre comparables, il est nécessaire d'obtenir des segments plus longs, caractéristiques d'un mouvement plus global : une compression des trajectoires doit être entreprise. On utilisera pour cela le principe MDL avec la formalisation proposée par la méthode de TRACLUS. Ensuite, afin de pouvoir procéder à une extraction séquentielle de connaissance sur nos données, nous devons les discréteriser, c'est-à-dire diviser notre plage de valeurs quasi continues en un nombre fini de valeurs que l'on pourra comparer. Pour cette étape, nous appliquerons un clustering sur les segments composant nos trajectoires compressées. Après cette étape de discréétisation, des clusters de segments sont considérés comme équivalents. Il s'agit alors d'effectuer un recodage des trajectoires à partir des identifiants de ces clusters pour obtenir des trajectoires discréétisées. Nous pouvons finalement procéder à l'extraction de connaissances séquentielle sur les trajectoires.

2.1 Pipeline

Ce plan de traitement est illustré par le pipeline à la figure 2.1.

Appliqué à notre cas, nous obtenons les étapes suivantes :

1. **compression MDL** : le processus de compression implique la conversion de fichiers CSV représentant la position continue (trajectoire) des joueurs lors d'une partie en un fichier CSV contenant des points caractéristiques, qui forment la représentation compressée des trajectoires des joueurs. Nous appliquons cette méthode sur toutes nos parties.
2. **discréétisation par clustering** : de ces représentations compressées, nous récupérons les segments caractéristiques de nos trajectoires (sur toutes les parties à analyser), sur lesquels nous appliquons notre clustering. Nous écrivons le résultat dans un CSV.
3. **recodage** : à cette étape, nous avons des clusters de segments, représentants nos items, et nos trajectoires compressées. Nous recodons alors nos trajectoires compressées en une suite d'items correspondants dans un nouveau CSV, pour chaque partie. Cela permet de réintroduire la notion de temporalité.

4. **Extraction séquentielle** : Nous pouvons alors procéder à une extraction séquentielle, en utilisant l'algorithme PrefixSpan adapté à l'extraction séquentielle. Pour cette étape, nous pouvons découper nos trajectoires pour cibler différentes temporalités tactiques des parties.

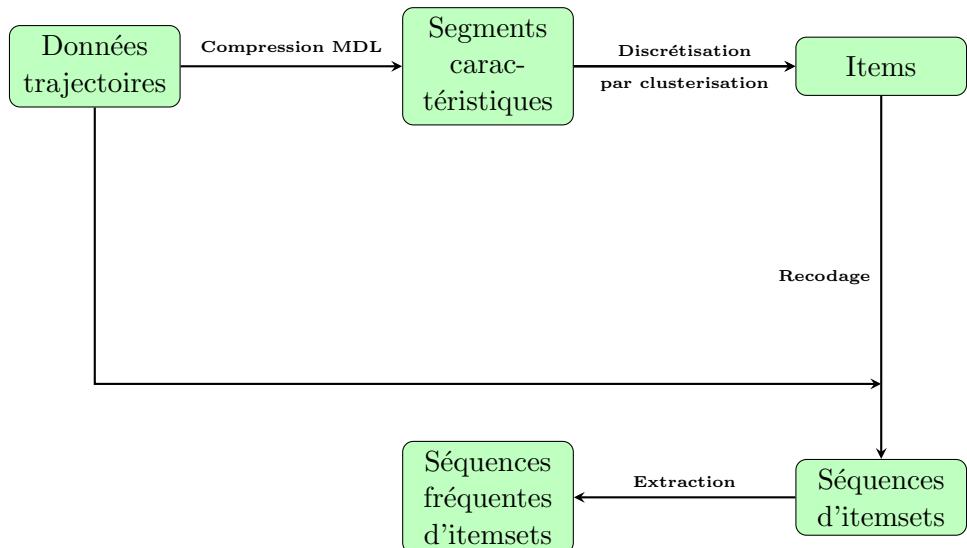


Figure 2.1: Illustration du pipeline.

Chapter 3

Réalisation du pipeline

3.1 Compression de trajectoire

Pour permettre une analyse des données par les algorithmes en un temps raisonnable, la compression est une étape importante du traitement. Cependant, il y a un équilibre à trouver entre concision et précision. Pour y arriver, il a été décidé d'adopter le principe MDL (Minimum Description Length) et plus particulièrement, une méthode permettant de trouver une solution approximative en temps linéaire.

Nous nous basons ici sur le travail de Jae-Gil Lee et Kyu-Young Whang [2].

Pour réaliser cette compression, il faut introduire la notion de distance à choisir, quelles sont les propriétés désirables pour une compression de trajectoire, la formalisation à l'aide du principe MDL. Avant de terminer sur la solution approximative proposée. Cette méthode de compression rapide est nécessaire pour traiter de grandes quantités de données de trajectoires tout en préservant une précision suffisante pour l'analyse ultérieure.

3.1.1 Propriétés désirables pour la compression de trajectoire

L'objectif est de trouver des points caractéristiques tels qu'ils se trouvent à clés dans la représentation de la trajectoire, notamment lors de changements flagrants de direction. Avec l'idée d'un équilibre entre précision et concision. Comme illustré ci-dessous 3.1.

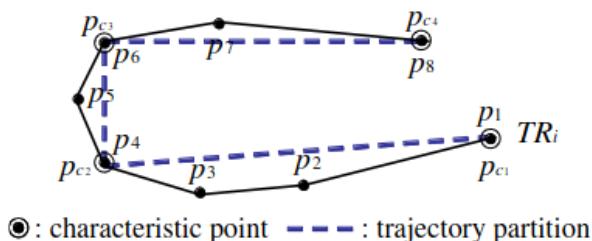


Figure 3.1: Exemple d'une trajectoire et de sa compression
Source : Trajectory Clustering: A Partition-and-Group Framework

3.1.2 Fonction de distance

Pour réaliser du clustering sur des segments de trajectoires, Jae-Gil Lee et Kyu-Young Whang ont défini une notion de distance composée de trois composantes comme suit : la distance perpendiculaire (d_{\perp}), la distance parallèle (d_{\parallel}) et la distance angulaire (d_{θ}). Elles sont illustrées de manière intuitive dans la Figure 3.2. Il est possible de voir le détail des calculs en annexe A.1.

Supposons que nous ayons 2 segments en n dimensions, $L_i = s_i e_i$ et $L_j = s_j e_j$, où s_i, e_i, s_j et e_j représentent des points de n dimensions. Nous assignons le segment plus long à L_i et le segment plus court à L_j .

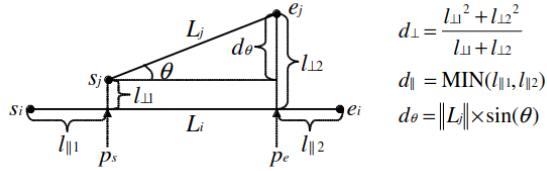


Figure 3.2: Les 3 composantes de distance entre 2 segments
Source : Trajectory Clustering: A Partition-and-Group Framework

3.1.3 Principe MDL formalisé

Le concept de Longueur de Description Minimum (MDL) est largement utilisé en théorie de l'information pour décrire un ensemble de théories et d'applications. L'idée clé de ce concept est que pour un ensemble de données (D) donné, la meilleure hypothèse (H) est celle qui conduit à la meilleure compression pour ces données. Les deux composantes de base de MDL sont : $L(H)$ et $L(D|H)$, où $L(H)$ est la longueur, en bits, de la description de l'hypothèse et $L(D|H)$ est la longueur, en bits, de la description des données lorsqu'elles sont encodées à l'aide de l'hypothèse. La meilleure hypothèse H pour expliquer D est celle qui minimise la somme de $L(H)$ et $L(D|H)$. Ici, $L(H)$ mesure le degré de concision et $L(D|H)$ celui de l'exactitude.

En formalisant ce concept ici :

$$L(H) = \sum_{j=1}^{p-1} \log_2(\text{len}(pc_j pc_{j+1})) \quad (3.1)$$

$$L(D|H) = \sum_{j=1}^{p-1} \sum_{k=c_j}^{c_{j+1}-1} \log_2(d_{\perp}(pc_j pc_{j+1}, pk pk_{k+1})) + \log_2(d_{\theta}(pc_j pc_{j+1}, pk pk_{k+1})) \quad (3.2)$$

3.1.4 Méthode de compression rapide en temps linéaire

La méthode proposée consiste à considérer que l'ensemble des optima locaux est l'optimum global. Cela s'apparente à une observation de l'évolution de l'entropie locale, ici l'équilibre local entre le $L(H)$ et $L(D|H)$, en sélectionnant à chaque étape le point suivant comme point caractéristique. Si l'entropie augmente brusquement, il faut alors sélectionner le point précédent comme point caractéristique et

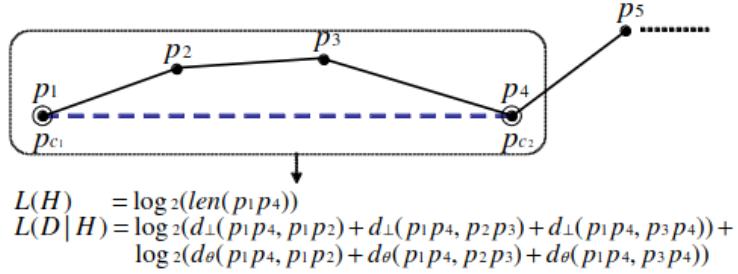


Figure 3.3: Exemple de la formalisation MDL sur un bout de trajeciores
 Source : Trajectory Clustering: A Partition-and-Group Framework

reprendre à partir de celui-ci. Cela s'illustre bien sûr la figure 3.1. Voir annexe pour plus de détails A.2.

3.2 Clustering

Le clustering 3.4 est une technique d'analyse de données qui permet de regrouper des données similaires ensemble en des groupes appelés "clusters". Le but du clustering est d'identifier des structures dans les données qui ne sont pas évidentes à observer à première vue.

Le clustering est une technique non supervisée, ce qui signifie qu'il n'y a pas de variable de réponse prédéfinie. Au lieu de cela, les algorithmes de clustering examinent les caractéristiques des données et les regroupent en fonction de leur similarité. Les données qui sont similaires sont regroupées dans le même cluster, tandis que les données qui sont différentes sont regroupées dans des clusters différents.

Il existe plusieurs types d'algorithmes de clustering, chacun avec ses propres avantages et inconvénients. Le choix de l'algorithme de clustering dépendra des données et de l'objectif de l'analyse. Les clusters peuvent être utilisés pour découvrir des tendances ou des relations cachées dans les données, ou pour segmenter les clients en fonction de leurs caractéristiques communes.

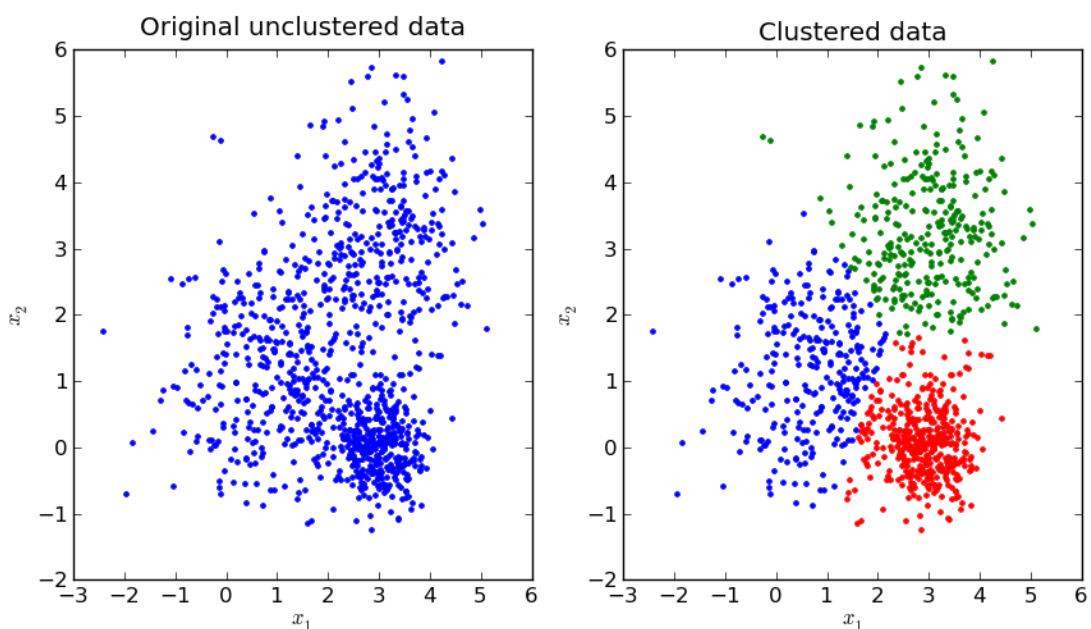


Figure 3.4: Exemple de clustering
 Source : mubaris clustering

3.2.1 K-Means et K-Medoids

Fonctionnement de K-Means:

1. spécifier le nombre de clusters(K).
2. Ensuite, sélectionner K éléments de données initiaux qui serviront de centres de cluster. Ces éléments peuvent être choisis au hasard ou selon un critère spécifique.
3. À présent, l'algorithme attribue chaque élément de données au cluster le plus proche en termes de distance. Cela signifie que chaque élément est assigné au centre de cluster le plus proche.
4. Une fois que tous les éléments de données ont été attribués à un cluster, les centres de cluster sont recalculés en calculant la moyenne de tous les éléments qui leur ont été assignés.
5. Les étapes 3 et 4 sont répétées jusqu'à ce que les centres de cluster ne changent plus, ou que le nombre d'itérations maximum soit atteint.

À la fin du processus, chaque élément de données appartient à un cluster spécifique, et chaque cluster est caractérisé par son centre. L'algorithme K-means vise à minimiser la somme des distances entre chaque élément de données et son centre de cluster correspondant. C'est ce qu'on appelle la fonction objective de l'algorithme.

L'algorithme K-means est rapide et efficace pour de petits et moyens ensembles de données. Toutefois, il peut être sensible aux éléments de données initiaux sélectionnés au hasard, et il peut avoir du mal à trouver la solution optimale pour de grands ensembles de données ou pour des données ayant une structure complexe.

L'algorithme k-medoids est sensiblement similaire à celui de k-means. La différence étant qu'au lieu de prendre la moyenne du cluster comme centre, il faut prendre l'élément dont la somme des distances par rapport à tous les autres éléments du cluster est la plus petite. Cela permet de rester plus proche des données réelles. Mais cela représente un coût supplémentaire.

3.2.2 Propagation d'affinité

La propagation d'affinité est un algorithme de clustering proposé par Brendan J. Frey et Delbert Dueck, basé sur des données qui utilise une approche de propagation de similarité entre les points de données pour former des clusters. L'algorithme de propagation d'affinité utilise donc des mesures de similarité entre les points de données pour déterminer les clusters. L'implémentation est décrite en détail dans l'annexe B.1.

1. Initialisation: initialiser quatre matrices, une pour la similarité, une pour les responsabilités, une pour les disponibilités et une matrice finale qui va nous permettre d'extraire les différents clusters. Ces matrices sont de taille $N*N$ avec N , la taille de l'ensemble d'éléments.
2. Calcul de la similarité: calculer la similarité entre chaque paire d'élément de données en utilisant une mesure de similarité, pour ensuite remplir la matrice de similarité. Dans notre cas, nos éléments sont dotés d'une méthode afin de quantifier la distance d'un élément à un autre. L'intérêt de cette matrice est de quantifier à quel degré un élément est semblable à un autre.

3. Calcul des responsabilités: l'algorithme utilise ensuite les valeurs de la similarité pour calculer les responsabilités de chaque élément en tenant compte des valeurs de leur disponibilité. Les responsabilités mesurent l'importance d'un élément pour les autres points de données.

$$R(i, k) \leftarrow S(i, k) - \max_{k' \neq k} \{D(i, k') + S(i, k')\} \quad (3.3)$$

Figure 3.5: Formule de la responsabilité

4. Calcul des disponibilités: l'algorithme va ensuite utiliser la matrice des responsabilités pour mettre à jour la matrice des disponibilités. Les disponibilités mesurent l'importance des autres points de données pour un élément donné.

$$D(i, k) \leftarrow \min \left\{ O, R(k, k) + \sum_{i' \neq i, i' \neq j} \max \{0, R(i', k)\} \right\} \quad (3.4)$$

Figure 3.6: Formule de la disponibilité

5. Assignation des clusters: les étapes 3 et 4 vont être réitérées un nombre n de fois dans notre implémentation. Notre algorithme va ensuite venir mettre à jour la matrice finale, souvent appelée matrice de "critère" ou de "préférence", elle assigne chaque élément à un cluster en fonction de la responsabilité maximale de chaque élément. Cette matrice se calcule en faisant le somme matricielle de de responsabilité et de la disponibilité.

Quels sont les inconvénients d'un tel algorithme ?

L'inconvénient principal de cet algorithme est sa complexité quadratique $O(n^2)$. L'algorithme de propagation d'affinité nécessite une grande quantité de mémoire pour stocker ses matrices, qui sont des matrices carrées de la taille de l'ensemble au carré. Pour de grands ensembles de données, cela peut nécessiter une quantité considérable de mémoire, ce qui peut rendre l'exécution de l'algorithme difficile, voire impossible sur des ordinateurs avec une mémoire limitée.

Le temps d'exécution est lui aussi impacté par la taille du jeu de donnée à traiter.

Quels en sont les avantages ?

Bien que la complexité de cet algorithme puisse réduire le nombre d'applications de cet algorithme, son principal avantage est que contrairement aux méthodes de clustering traditionnelles, propagation d'affinité n'a pas besoin de connaître le nombre de clusters à assigner, il va automatiquement le trouver tout en gardant une certaine pertinence des résultats.

Malgré sa complexité, les calculs des matrices de cet algorithme sont parallélisables, rendant le calcul de celles-ci moins contraignantes sur de grands jeux de données.

Une autre optimisation de cet algorithme est d'implémenter les itérations en calcul matriciel au lieu d'utiliser des boucles, rendant le calcul bien moins couteux en temps.

sources :

Rapport de Brendan J. Frey et Delbert Dueck
Implémentation faite par la librairie scikit-learn
Publication de Towards Data Science

3.3 Extraction de connaissance dans les données

L'extraction de connaissance est le processus d'extraction d'informations utiles et exploitables à partir de sources de données brutes et non structurées, telles que des documents, des bases de données, des fichiers texte, des images, des vidéos et d'autres types de médias numériques.

Les principales tâches de l'extraction des données sont le **clustering**, l'analyse des valeurs **aberrantes**, la **classification** et l'**extraction de motif fréquents**.

3.3.1 Extraction séquentielle

L'extraction séquentielle est une technique d'analyse de données qui consiste à identifier des motifs séquentiels récurrents dans une série temporelle ou une séquence d'événements.

Plus précisément, l'extraction séquentielle recherche des sous-séquences qui se produisent fréquemment dans une séquence donnée et les identifie comme des motifs significatifs. Ces motifs peuvent ensuite être utilisés pour comprendre le comportement de la séquence, pour prédire des événements futurs, pour détecter des anomalies ou pour d'autres applications.

Par exemple, si l'on analyse les achats d'un client sur une période, l'extraction séquentielle pourrait identifier des motifs tels que "le client achète toujours du café après avoir acheté du lait". Cela pourrait aider une entreprise à mieux comprendre les préférences du client et à proposer des offres ciblées.

En résumé, l'extraction séquentielle est une technique utile pour trouver des modèles récurrents dans une série temporelle ou une séquence d'événements, ce qui peut aider à mieux comprendre le comportement et à prendre des décisions éclairées en conséquence.

Exemple

SID	Sequence
1	$\langle \{a, b\}, \{c\}, \{f, g\}, \{g\}, \{e\} \rangle$
2	$\langle \{a, d\}, \{c\}, \{b\}, \{a, b, e, f\} \rangle$
3	$\langle \{a\}, \{b\}, \{f, g\}, \{e\} \rangle$
4	$\langle \{b\}, \{f, g\} \rangle$

Figure 3.7: Exemple de séquence d'items
source : The data mining blog

Dans cet exemple 3.7, les données en question contiennent quatre séquences de transactions d'achats effectuées par différents clients. Chaque séquence représente les articles achetés par un client à différents moments. Une séquence est une liste ordonnée d'ensembles d'articles achetés ensemble, appelés itemsets.

Dans cet exemple 3.7, la première séquence (identifiée par SID 1) indique que le client a acheté des articles a et b ensemble, puis a acheté l'article c seul, ensuite a acheté les articles f et g ensemble, puis a acheté l'article g seul, et enfin a acheté l'article e seul.

Dans cet exemple 3.7, nous pouvons noter que le pattern séquentiel $\langle\{a\},\{f,g\}\rangle$ qui apparaît dans deux transactions une et deux, elle est donc un motif fréquent avec un support de 2, le support étant le nombre de transactions contenant cette séquence. De même, nous pouvons voir le pattern $\langle\{a\},\{g\}\rangle$ est un motif fréquent avec un support de 3, car apparaissant dans les transactions 1, 2 et 3.

Dans notre situation, nous cherchons donc des motifs fréquents qui illustreront les déplacements des joueurs lors des games. Par exemple, si le pattern $\langle\{A\},\{F\},\{R\}\rangle$ exprimant les positions du joueur est motif qui apparaît de manière fréquente lors des games, il exprimera le fait que si le joueur se trouve (appartient) à un moment T au cluster A il ira ensuite au cluster F, puis B avec un certain niveau de confiance, avec la confiance représentant la probabilité de réalisation.

Chapter 4

Expérimentations et Discussion

4.1 Compression

Les résultats obtenus lors de la compression illustrent l'efficacité de la méthode proposée pour identifier les points caractéristiques d'une trajectoire. En effet, cette méthode nous permet d'atteindre une réduction moyenne de 67.9% de toutes nos trajectoires. Ces résultats soulignent l'importance de notre approche pour la compression de trajectoires, tout en maintenant leur qualité, comme illustré ici 4.1.

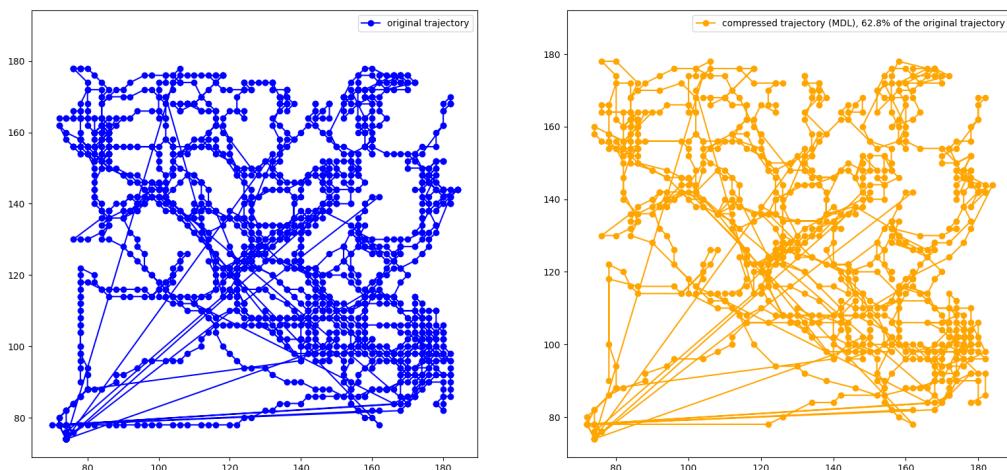


Figure 4.1: Exemple d'une trajectoire avant et après compression

Il est possible d'observer plus finement l'équilibre entre concision et précision que permet cette méthode sur la figure suivante 4.2.

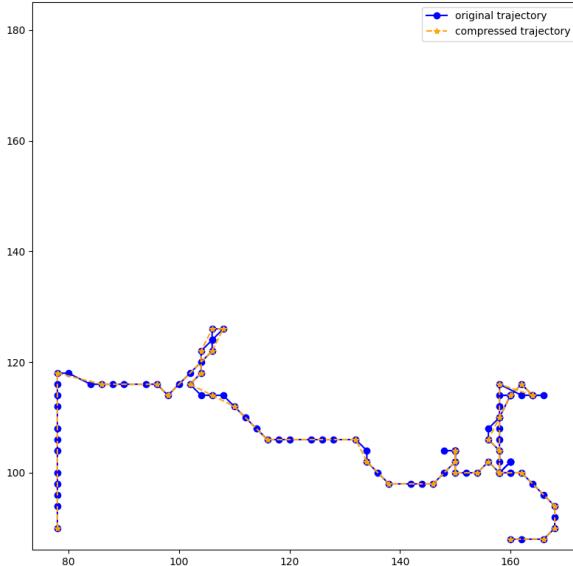


Figure 4.2: Exemple de la compression sur les 100 premiers points d'une trajectoire

4.2 Clustering

Pour réaliser un clustering de qualité, il faut d'abord se pencher sur l'étude des paramètres existants. Dans notre cas, il s'agira de l'évaluation de la distance entre deux segments orientés et du nombre de clusters.

Afin d'illustrer plus parlant, il a été décidé de se concentrer uniquement sur les trajectoires de l'équipe de gauche.

Note : Les résultats obtenus avec la méthode de propagation d'affinité présentent un biais significatif en raison de sa complexité spatiale et temporelle, qui suit une complexité en $O(n^2)$. Ainsi, même après compression, le volume important de segments rend difficile son exécution pour plus de deux parties.

4.2.1 Paramétrisation de la distance entre segments

La distance utilisée est la somme pondérée de trois distances : distance angulaire, distance perpendiculaire et distance parallèle. Les coefficients de cette somme ayant un impact significatif sur les résultats de cette partie, il est nécessaire d'évaluer, de trouver le paramétrage le plus pertinent.

Pour cette partie, il a été choisi de fixer le nombre de clusters à 100 pour k-means et k-medoids. Il a été remarqué que cela n'avait pas d'impact particulier sur le paramétrage de la distance.

Pour y arriver, plusieurs expérimentations ont été menées successivement afin d'observer les variations qu'impliquent les paramètres pour chaque algorithme. Une fois ce tour d'horizon réalisé, il est possible d'orienter les choix plus précisément.

k-means

Sur ces résultats 4.3, il est possible de distinguer nettement les différences existantes entre chacune des paramétrisations. Lorsque son facteur est égal à 0, la distance angulaire est critique pour obtenir

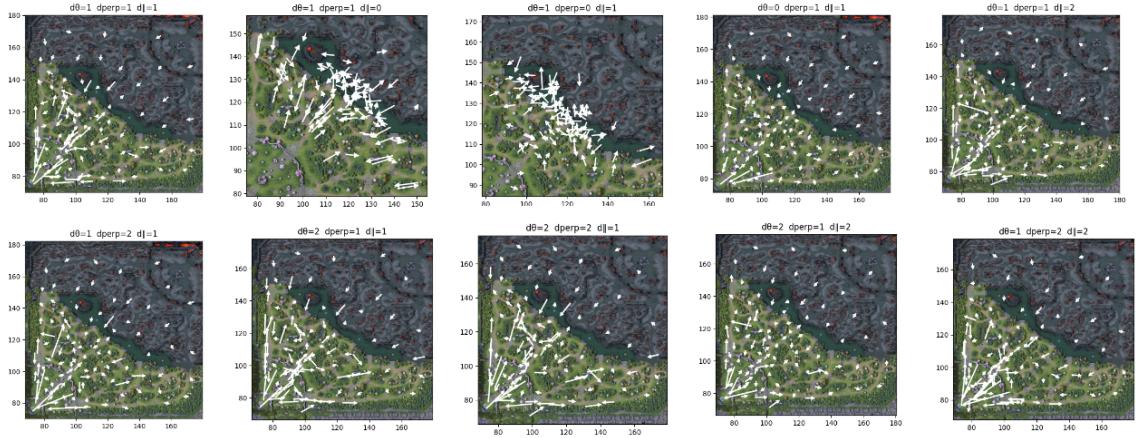


Figure 4.3: Résultats clustering k-means pour différentes paramétrisations de la distance

des résultats couvrant correctement les trajectoires sur toute la carte. C'est également le cas pour la distance perpendiculaire. Concernant la distance parallèle, il semblerait qu'elle permette d'affiner le groupement de segments de même orientation et obtenir des segments représentatifs plus longs sur les morceaux de trajectoire récurrents. Cet équilibrage est ici le plus pertinent avec l'adoption d'un facteur 1/2 pour la distance parallèle.

En partant de ces observations, il est apparu qu'un facteur de 1/5 pour la distance parallèle et de 1 pour les deux autres distances donnait des résultats satisfaisants Comme illustré ici 4.4.

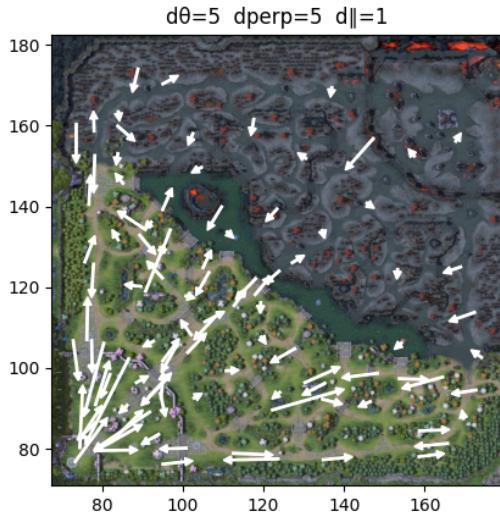


Figure 4.4: Résultat clustering k-means avec un facteur parallèle de 1/5

k-medoid

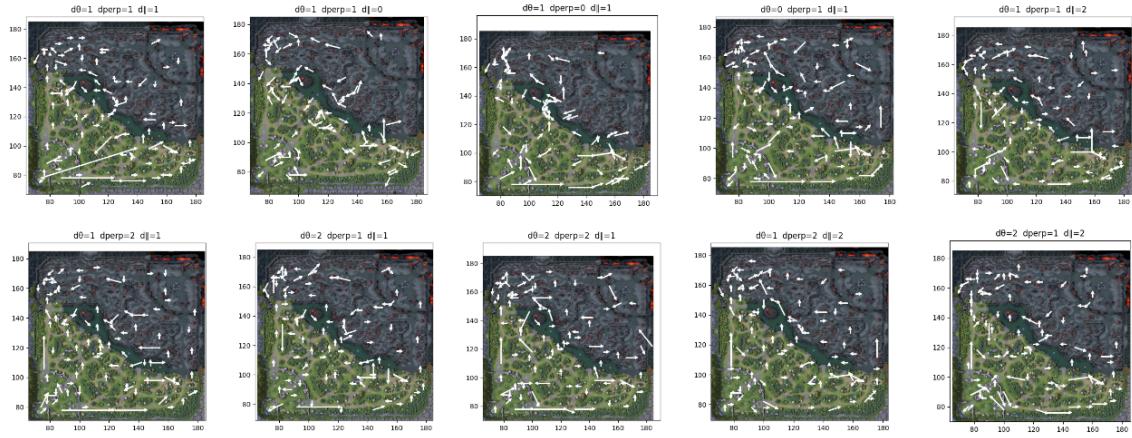


Figure 4.5: Résultats clustering k-medoids pour différentes paramétrisations de la distance

Les résultats des tests 4.5 ont montré que le clustering effectué par k-medoid reste relativement constant. Les trois distances jouent un rôle crucial et aucune des trois ne vraiment être négligée. La distance parallèle et perpendiculaire jouent le rôle le plus important en permettant une distribution plus homogène des clusters sur la carte. En effet, elles permettent la prise en compte de trajectoires représentants des mouvements directes sur une durée plus grande.

La distance angulaire permet une meilleure différentiation des trajectoires de direction proches.

Au vu de ces résultats, il semble que partir sur un quasi-équilibre entre les trois distances est la meilleure stratégie à adopter afin d'obtenir les meilleurs résultats 4.6.

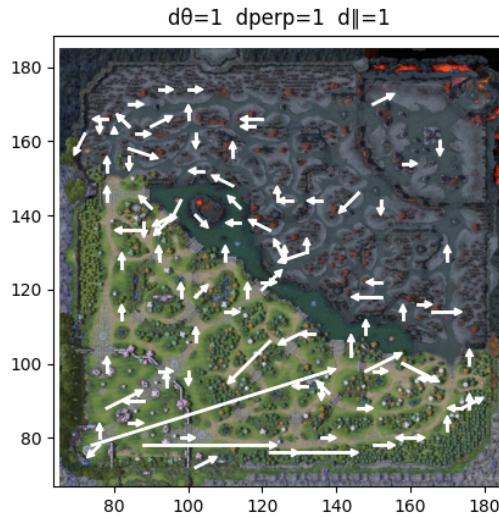


Figure 4.6: K-medoids avec un paramétrage de 1 1 1

propagation d'affinité

Pour propagation d'affinité, les résultats 4.7 sont globalement assez similaires et traduisent davantage une notion de position que de partitions de trajectoire. En effet, les segments sont généralement courts

et répartis sur l'ensemble de la carte de façon homogène. De plus, il y a une tendance à obtenir des segments caractéristique qui sont verticaux, horizontaux ou diagonal.

Néanmoins, il faut prendre en compte que ces résultats sont sujets au biais du faible nombre de parties. De la même façon, cet algorithme possède des paramètres propres, comme le damping, mesure d'acceptance, qui vient affecter le nombre de clusters trouvé.

Ici, il a été choisi d'utiliser un damping de 0.9 avec 60 itérations pour chaque expérimentation.

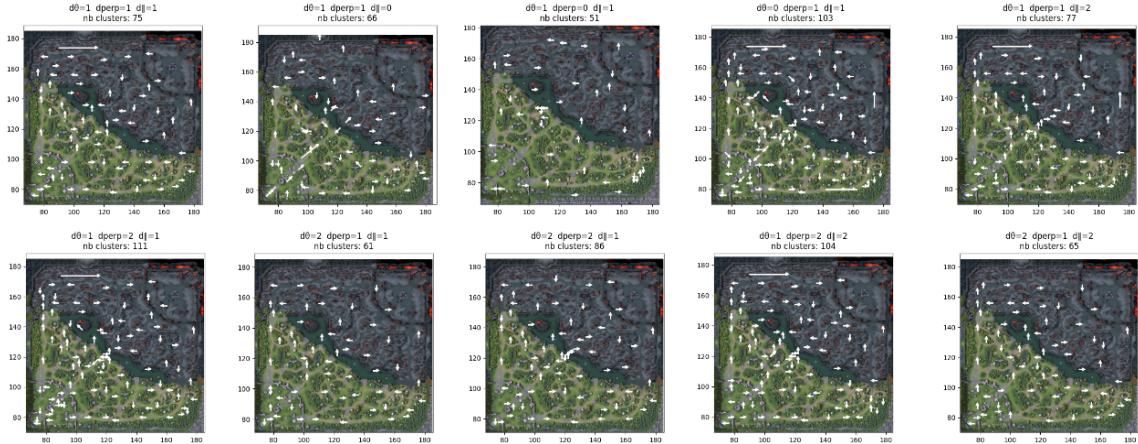


Figure 4.7: Résultats clustering propagation d'affinités pour différentes paramétrisations de la distance

Au vu de ces résultats, le plus pertinent reste la mise à 0 du facteur d'angle, comme sur la figure 4.8.

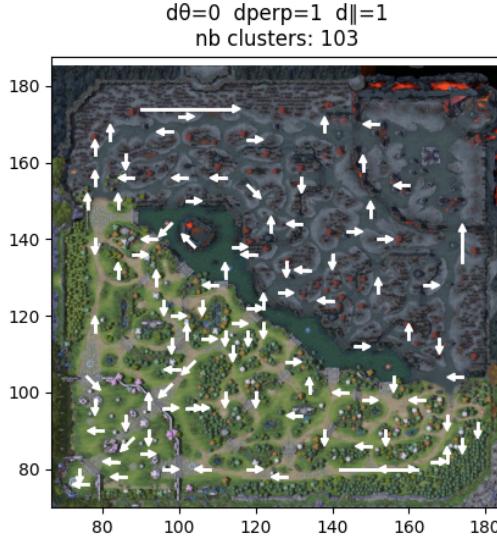


Figure 4.8: Résultat clustering propagation d'affinité avec un facteur d'angle de 0

Discussion

Il est notable que la paramétrisation a des effets différents sur chacune des méthodes. Cela est assez étonnant dans le cas de k-means et k-medoids, mais la différence s'explique par la différence fondamentale entre les deux algorithmes, ce qui donne aussi sa qualité à k-medoids.

Il est cependant difficile de les comparer avec propagation d'affinités, car celui-ci va également faire

une estimation du nombre correct de clusters, en plus de son biais initial lié à son nombre de parties traitées.

Également, il est intéressant d'observer les différences dans les représentations des trajectoires de chaque côté de la carte, avec des petits segments dans la moitié de carte ennemi (droite), et de longs segments orientés à gauche. Les courts segments semblent représenter des positions sur un espace de la carte 4.9. Tandis que les segments orientés allongés semblent représenter des trajectoires fréquentes avec une direction précise 4.10.



Figure 4.9: Illustration d'un représentant jugé "positionnel"



Figure 4.10: Illustration d'un représentant de trajectoires

En terme d'information de jeu, il est possible d'interpréter ces différences. Quant aux trajectoires possibles de chaque côté de la carte, cela s'explique pour deux raisons : premièrement, il est normal pour une équipe de rester de son côté de la carte. Deuxièmement, cela peut s'expliquer par le fait que les excursions du côté adverse de la carte se font moins souvent et sous la contrainte des déplacements des adversaires, ce qui les rend moins réguliers et moins précis d'une partie à l'autre.

4.2.2 Paramétrisation du nombre de clusters

Les algorithmes de k-clusters tels que k-means et k-medoids ont un inconvénient notable : ils nécessitent la spécification préalable du nombre de clusters souhaités, ce qui peut entraîner des résultats insatis-

faisants en raison de la difficulté à déterminer le nombre optimal de clusters et à choisir une discréétisation adaptée. Bien que l'algorithme de propagation d'affinité soit une alternative intéressante à ce problème, il est difficile de comparer ses résultats avec ceux des autres algorithmes, en raison des différences dans la paramétrisation des distances et du nombre de clusters, comme l'ont montré les résultats précédents.

Il a été choisi de prendre les meilleurs paramètres de distance pour chaque algorithme pour obtenir une évaluation plus intéressante pour chacun de ces cas.

k-means

Au vu des résultats de la figure 4.11, il est visible qu'à partir de 200 clusters, plusieurs représentants se chevauchent, rendant la discréétion moins intéressante. Cependant, un bon équilibre semble atteint pour un nombre de 150 clusters (figure 4.11b), où il est possible de suivre les trajectoires fréquentes sans que la carte, sans observer trop de chevauchements sur représentants.

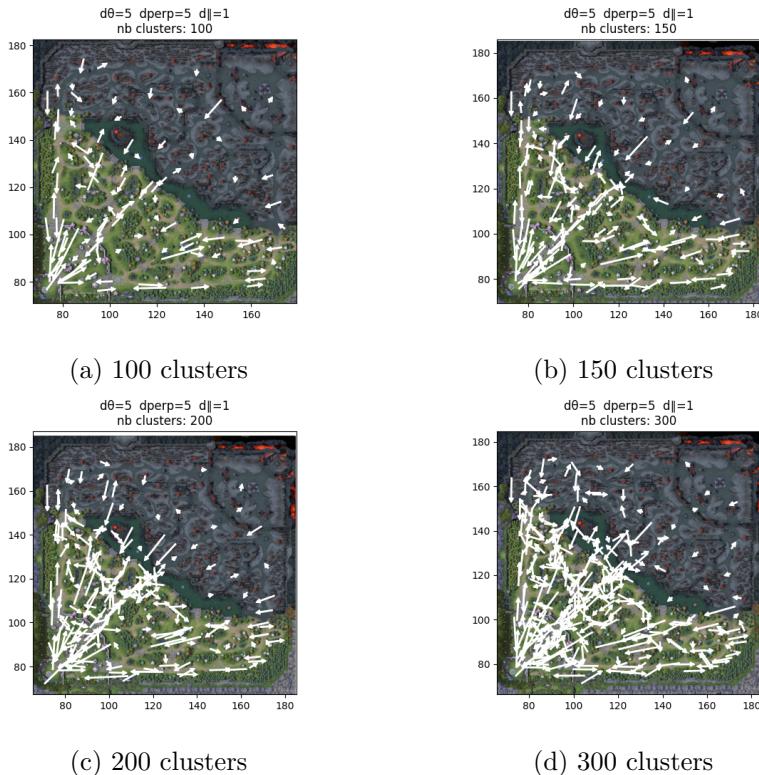


Figure 4.11: Évolution de k-means en fonction du nombre de clusters

k-medoids

Les résultats 4.12 montrent que la qualité de l'information tirée des clusters s'améliore avec leur nombre, mais au-delà de 200 clusters, la quantité d'informations devient trop importante pour en tirer des conclusions. Ainsi, un nombre de 200 clusters semble être un compromis acceptable pour k-medoids afin de suivre l'évolution des trajectoires et détecter les zones à forte fréquentation.

propagation d'affinité

La particularité de propagation d'affinité est justement d'évaluer le nombre de clusters idéal par lui-même. Sur ses meilleurs résultats (figue 4.8), nous avons obtenu un nombre de 103 clusters. Cependant,

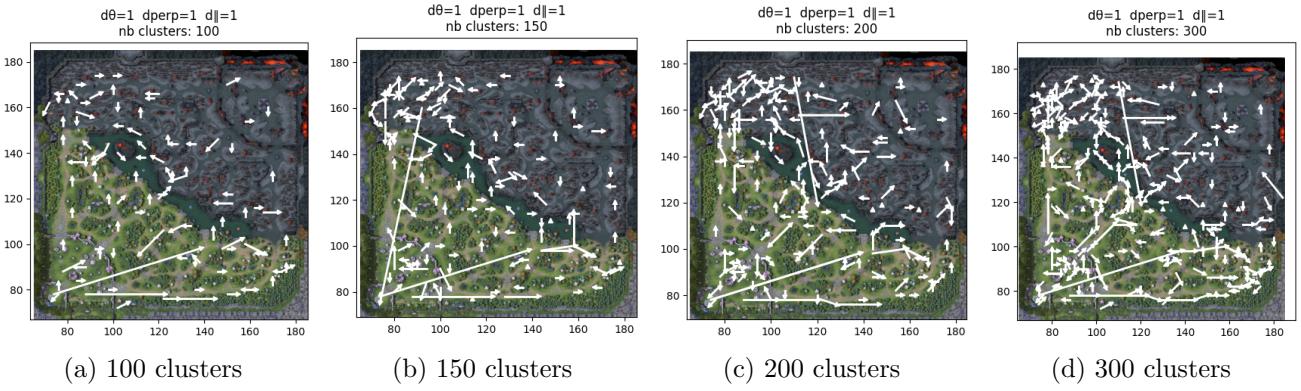


Figure 4.12: Évolution de k-medoids en fonction du nombre de clusters

il faut rappeler que ce résultat est aussi lié au facteur de damping et au nombre d’itérations, que nous n’avons pas fait varier.

Discussion

Là encore, les résultats optimaux pour chacun des algorithmes différent. Cela s’explique par leurs différences de paramétrage qui vont influer sur leur définition de représentants.

Malgré tout, concernant k-means et k-medoids, cela illustre d’autant plus leur différence de fonctionnement, en effet, la capacité de k-medoid à converger qualitativement sur des trajectoires fréquentes font qu’il est possible d’affiner la discréétisation sans brouiller nos résultats.

L’idéal aurait été d’utiliser la capacité de propagation à trouver le nombre correct de clusters, mais cela s’avère difficile ici. En effet, il aurait fallu que les algorithmes possèdent un paramétrage optimal équivalent. De plus, il aurait fallu à la fois surmonter la contrainte matérielle pour utiliser pleinement propagation d’affinité et mener une étude sur le facteur optimal de damping à utiliser.

4.3 Extraction séquentielle

Paramètres de l’extraction

L’algorithme d’extraction séquentiel utilisé [1] permet l’extraction de motifs fréquents. Plusieurs paramètres rentrent alors en compte.

1. **minsup(minimum support)** : le nombre minimum de transactions où doivent apparaître un motif fréquent.
2. **min_time_interval** : le temps minimal permis séparant deux itemsets successifs dans un motif.
3. **max_time_interval** : le temps maximal permis séparant deux itemsets successifs dans un motif.
4. **min_whole_interval** : le temps minimal permis séparant le premier itemset et le dernier itemset d’un motif fréquent (taille minimale du motif).
5. **max_whole_interval** : le temps maximal permis séparant le premier itemset et le dernier itemset d’un motif fréquent (taille maximale du motif).

4.3.1 Résultats

Concernant l'extraction, beaucoup de paramétrage ont été testés, mais malgré tout, peu de résultats concluants ont pu être obtenus dans le temps imparti.

Pour commencer, un découpage arbitraire a été expérimenté. De 30 à 120 secondes, pour finalement rester sur une moyenne de 60 secondes.

Dès lors, nous avons eu des problèmes d'extraction n'ayant pas spécialement d'intérêt où nous avions des morceaux de trajectoires qui ne se suivaient pas du tout et provenant très probablement de 2 joueurs différents, comme illustré sur la figure 4.14a, il a été décidé de se concentrer sur la trajectoire d'un joueur à la fois.

Ensuite, après les obtentions de résultats illustrant très largement des suites successives de la même trajectoire, nous avons voulu faire varier l'intervalle de temps entre chaque transaction étudié dans l'idée d'éviter le cas où la vitesse ou le rythme des positions des joueurs sur chaque morceau de trajectoire rendrait impossible l'extraction de motifs d'une taille supérieure à 2. Comme illustré sur la figure 4.13. Même si cela pourrait engendrer une perte d'information, il a quand même été jugé nécessaire.

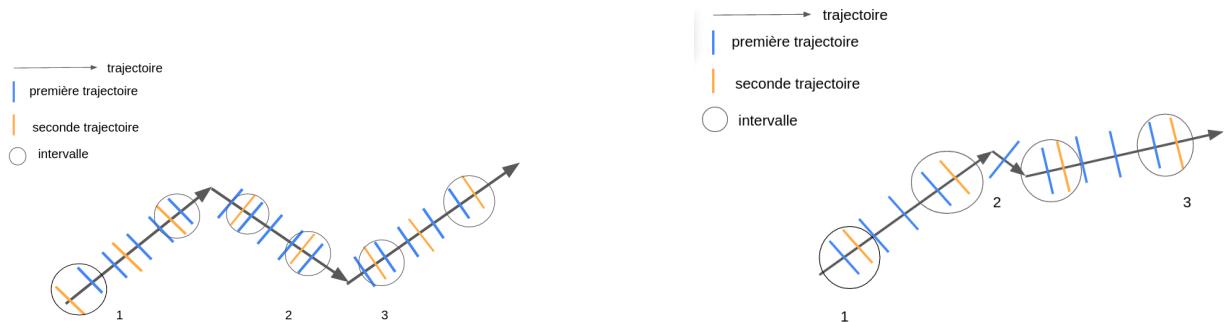


Figure 4.13: intervalle de temps

À ce moment-là, il a été possible d'extraire des motifs formés d'une suite de morceau de trajectoire, exemple figure 4.14b, mais ces résultats n'ont pu être obtenus qu'avec un support minimum assez faible (5%), ce qui n'était pas très satisfaisant. Cependant, il n'était toujours pas possible d'extraire des motifs composés de plus de 2 segments.



Figure 4.14: Résultat extraction

4.3.2 Discussion

Les résultats obtenus ne sont pas complets et mériraient d'aller plus loin et d'étudier plus amplement les paramètres disponibles ainsi que le choix du découpage. Par exemple, il serait intéressant d'essayer une extraction sur différents moments des parties (début, milieu de jeu, etc...) afin d'observer directement en fonction de chaque partie les motifs fréquents et de mieux pouvoir les analyser.

De plus, il serait possible d'essayer d'autres méthodes pour extraire des motifs significatifs sans passer par découpage en trajectoire, mais en passant par une discréétisation de la position (exemple : découpage de la carte en une grille de taille n). Et d'ensuite réaliser une extraction sur ces points. L'avantage étant de ne pas passer par l'étape de recodage ni de la compression sous forme de trajectoire et de privilégier l'information positionnelle.

Chapter 5

Conclusion

En conclusion, ce projet a permis de mettre en évidence la méthode d'analyse de donnée et plus particulièrement les enjeux liés à l'analyse de trajectoire. Ici des trajectoires issues d'un jeu vidéo. L'objectif étant de découvrir les trajectoires fréquentes et les motifs de jeu récurrents. Pour y arriver, un pipeline 2.1 a été établi pour réaliser le traitement nécessaire à l'exploitation de ces données. La problématique étant d'explorer différentes méthodes de discréétisation et mettre en avant les différences de ces dernières. Tout en gardant en tête l'objectif de discuter l'intérêt des régularités découvertes dans les données. Au cours du projet s'est imposé une problématique supplémentaire, celle de la paramétrisation de ces algorithmes et du modèle de représentation choisi a été abordé.

Ainsi, par une compression basée sur le principe MDL, et avec l'utilisation des algorithmes k-means, k-medoids ou propagation d'affinité pour la discréétisation, nous avons pu extraire des régularités intéressantes. Il a même été possible d'aller un peu plus loin avec une extraction séquentielle de motifs de jeu.

5.1 Résultats

Les données initiales étant continue, elles sont donc très volumineuses. Grâce à l'étape de compression 4.1, ce volume a été réduit en moyenne de 67.9%. Ceci en gardant un niveau idéal de précision et de concision.

Pour l'étape de discréétisation, il s'est avéré crucial de trouver un paramétrage approprié. Paramétrage d'abord sur le modèle de représentation sous forme de segments des morceaux de trajectoires. Puis sur le nombre de clusters à privilégier. Les résultats (figures 4.3, 4.5 et 4.7) ont conduit à la conclusion que les différents algorithmes de clustering - bien qu'ils partagent le même objectif - nécessitent différents paramétrages. Sur le nombre de clusters, bien qu'ayant un fonctionnement très proche, k-means 4.11 et k-medoids 4.12 n'ont pas le même nombre de clusters optimal. Il en va de même pour leurs spécifications de notion de distance. Cette dernière donnant contre toutes attentes des résultats bien différents sur une paramétrisation pour les deux algorithmes. Propagation d'affinités s'est montrée inadapté sur le volume de donnée traité. En effet, bien qu'il soit généralement très efficace et pertinent dans bien d'autre cas, ici sa complexité en $O(n^2)$ a été un facteur très limitant dans son étude.

En ce qui concerne la comparaison entre les différents algorithmes, les résultats ont montré que k-means et Propagation d'Affinité ont tendance à mettre l'accent sur le positionnement des joueurs 4.9a plutôt que sur l'évolution de leurs trajectoires, particulièrement dans le cas de la propagation d'affinité

4.8, tandis que k-medoids montre davantage l'évolution des trajectoires. Cette différence s'explique par les méthodes spécifiques utilisées par chaque algorithme pour effectuer le clustering.

Pour l'extraction séquentielle, les résultats obtenus sont partiellement incomplets et peu concluants en raison du manque de temps et de tests plus approfondis. Les motifs n'ont pu être extraits qu'avec un support faible et ne sont jamais composés de plus de deux segments successifs. Il convient toutefois de noter que le modèle utilisé pourrait ne pas être le plus adapté pour cette étude.

5.2 Améliorations

Sur ce projet, l'analyse séquentielle offre le plus grand potentiel d'amélioration significative. Le projet s'est principalement concentré sur l'étude d'une seule des deux équipes pendant les différentes parties. Il serait intéressant d'analyser les deux équipes simultanément en identifiant les actions de chaque équipe par rapport à l'autre et en extrayant des tactiques spécifiques à chacune d'elles. Alternativement, il serait pertinent d'analyser chaque équipe individuellement pour extraire les tactiques qu'elle utilise habituellement lors de différentes parties auxquelles elle a participé.

Il est également pertinent de mentionner la possibilité de suivre une voie différente que celle utilisée afin d'arriver au même objectif. Par exemple, la discrétisation pourrait être réalisée par un découpage direct de la carte pour avoir une notion de position étendu. Cela aurait pour avantage de garder de préserver un aspect spatio-temporel plus précis concernant les déplacements des joueurs. L'inconvénient serait d'obtenir au moment de l'extraction les trajectoires courantes comme motifs, mais il serait possible d'analyser plus finement l'organisation des équipes à des moments précis. De la même manière, avec une connaissance du jeu avancé, il serait possible de mener une étude précise sur les motifs concernant les objectifs du jeu.

Appendix A

Compression MDL

A.1 Fonction de distance

Pour réaliser du clustering sur des segments de trajectoires, Jae-Gil Lee et Kyu-Young Whang ont défini une notion de distance composée de trois composantes comme suit : la distance perpendiculaire (d_{\perp}), la distance parallèle (d_{\parallel}) et la distance angulaire (d_{θ}). Elles sont illustrées de manière intuitive dans la Figure A.1. Elles sont définies formellement de la façon suivante.

Supposons que nous ayons 2 segments en n dimensions, $L_i = s_i e_i$ et $L_j = s_j e_j$, où s_i, e_i, s_j et e_j représentent des points de n dimensions. Nous assignons le segment plus long à L_i et le segment plus court à L_j .

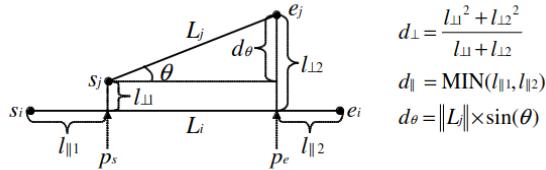


Figure A.1: Les 3 composantes de distance entre 2 segments
Source : Trajectory Clustering: A Partition-and-Group Framework

1. La distance perpendiculaire d_{\perp} :

$$d_{\perp}(L_i, L_j) = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}}, \quad (\text{A.1})$$

où $l_{\perp 1}$ est la distance euclidienne entre $\vec{s_j}$ et le point de projection p_s de $\vec{s_j}$ sur L_i , et $l_{\perp 2}$ est la distance euclidienne entre $\vec{e_j}$ et le point de projection p_e de $\vec{e_j}$ sur L_i .

2. La distance parallèle d_{\parallel} est définie comme :

$$d_{\parallel}(L_i, L_j) = \min(l_{\parallel 1}, l_{\parallel 2}), \quad (\text{A.2})$$

où $l_{\parallel 1}$ est la distance euclidienne minimale entre p_s et $\vec{s_i}$ ou $\vec{e_i}$, et $l_{\parallel 2}$ est la distance euclidienne minimale entre p_e et $\vec{s_i}$ ou $\vec{e_i}$.

3. La distance d'angle d_θ est définie comme :

$$d_\theta(L_i, L_j) = \begin{cases} \|Lj\| \times \sin(\theta), & \text{si } 0^\circ \leq \theta < 90^\circ \\ \|Lj\|, & \text{si } 90^\circ \leq \theta \leq 180^\circ \end{cases} \quad (\text{A.3})$$

où θ est l'angle d'intersection le plus petit entre L_i et L_j et $\|Lj\|$ est la longueur de L_j .

Nous définissons finalement la distance entre deux segments de droite comme suit :

$$\text{dist}(L_i, L_j) = w_\perp \cdot d_\perp(L_i, L_j) + w_\parallel \cdot d_\parallel(L_i, L_j) + w_\theta \cdot d_\theta(L_i, L_j)$$

Les poids w_\perp , w_\parallel et w_θ seront à déterminer en fonction des applications. Dans notre cas, nous les fixerons défaut à 1, car selon l'auteur ces valeurs fonctionnent généralement bien dans de nombreuses applications et ensembles de données.

A.2 Compression en temps linéaire

La méthode proposée consiste à considérer que l'ensemble des optima locaux est l'optimum global. On définira $MDLpar(p_i, p_j)$ le coût MDL ($L(H) + L(D|H)$) d'une trajectoire entre p_i et p_j (i,j) en supposant que p_i et p_j sont les seuls points caractéristiques. Et $MDLnopar(p_i, p_j)$ le coût MDL en supposant que tous les points de p_i à p_j sont caractéristiques, autrement dit, en préservant la trajectoire originale. Notons que dans $MDLnopar(p_i, p_j)$, $L(D|H)$ est nul. Un optimum local est la partition de trajectoire $p_i p_j$ la plus longue qui satisfait $MDLpar(p_i, p_k) \leq MDLnopar(p_i, p_k)$ pour tout k tel que $i < k \leq j$. Si le premier est inférieur au second, cela signifie que choisir p_k comme point caractéristique rend le coût de MDL plus petit que de ne pas le choisir.

L'algorithme fonctionne de la manière suivante :

1. Nous calculons $MDLpar$ et $MDLnopar$ pour chaque point d'une trajectoire.
2. Si $MDLpar$ est supérieur à $MDLnopar$, nous insérons le point précédent immédiat dans l'ensemble des points caractéristiques. Et nous répétons la même procédure à partir de ce point.
3. Sinon, nous augmentons la longueur d'une partition de trajectoire candidate. Et on continue

Appendix B

Propagation d'affinité

B.1 Implémentation

Comment avons-nous implémenté cet algorithme ?

Par soucis de lisibilité, notons :

- S, la matrice de similarité.
- R, la matrice de responsabilité.
- D, la matrice de disponibilité.

La structure de nos matrices est faite pour pouvoir comparer un élément d'index I à un élément d'index J, il est donc nécessaire d'utiliser une structure de liste ordonnée et fixe.

Pour commencer, il nous faut remplir la matrice de similarité. Chacun de nos objets donnés est capable au préalable de se comparer à d'autres objets du même type.

Une méthode distance nous permet donc de comparer un objet I à un objet J.

(Voir l'algorithme implémenté B.2)

Mettre à jour la matrice de responsabilité se fait une fois la similarité calculée. (voir figure 3.5)

De manière plus explicite, la responsabilité d'un élément i par rapport à k se calcule par sa valeur de similarité, moins la valeur maximale de la disponibilité pour les éléments i en considérant tous les autres points de données j, sauf l'élément k. De cette manière, nous pouvons quantifier l'importance d'un élément pour les autres éléments.

(Voir l'algorithme implémenté B.3)

Un paramètre que nous avons nommé "damping" est un paramètre d'amortissement compris entre 0 et 1 qui contrôle la quantité de propagation d'affinité entre les éléments. Nous l'appliquons à la formule de cette manière :

$$R(i, k) = R(i, k) * damping + S(i, k) - (1 - damping) * \max \{D(i, k') + S(i, k')\} \quad (\text{B.1})$$

Plus ce paramètre tend vers 0, plus les éléments similaires ne seront regroupés qu'avec leurs voisins

directs. Au contraire, plus le damping est proche de 1, plus les points de données similaires seront regroupés ensemble, indépendamment de leur position dans l'ensemble des éléments.

Mettre à jour la matrice de disponibilité se fait une fois la responsabilité mise à jour.
(Voir l'algorithme implémenté B.4)

$R(k,k)$ est l'élément de la matrice de responsabilité R correspondant à la similarité entre le point k et lui-même. La somme est prise sur tous les points de données i' autres que i et j , et mesure l'influence des autres points de données sur la disponibilité du point i pour être affecté au cluster contenant le point j . La fonction max est utilisée ici pour ne prendre en compte que les valeurs positives de la matrice de responsabilité, ne considérant donc que les éléments i' qui ont une similarité positive avec le point k . Enfin, la fonction min est là pour contraindre le résultat final à être négatif ou nul. Cela permet de quantifier l'importance des autres éléments pour un élément donné.

Une fois le processus de mise à jour de la responsabilité et la disponibilité répété un certain nombre de fois, nous finissons par faire l'addition matricielle de la matrice de responsabilité avec celle de disponibilité pour pouvoir en extraire les clusters et leur représentant, nous appelons cette matrice la matrice de critère. La valeur de critère la plus élevée de chaque colonne est désignée comme exemplaire et les colonnes qui partagent le même exemplaire se trouvent dans le même cluster.

Le processus de mise à jour de la responsabilité et la disponibilité sont exécutés un nombre arbitrairement de fois dans notre implémentation. Cependant, cet algorithme converge relativement rapidement, calculer la matrice de critère à chaque itération permettrait de détecter une convergence et ainsi éviter de réaliser des itérations inutiles.

B.2 Méthode de calcul de la similarité

```

1:  $E \leftarrow$  L'ensemble d'éléments
2:  $T \leftarrow$  la taille de  $E$ 
3:  $S \leftarrow$  Une matrice de taille  $T*T$ 
4: for  $I$  allant de 0 à  $T$  do
5:   for  $J$  allant de 0 à  $T$  do
6:      $S(I,J) \leftarrow E(I).distance(E(J))$ 
7:   end for
8: end for
```

B.3 Méthode de calcul de la responsabilité

```

1:  $E \leftarrow$  L'ensemble d'éléments
2:  $T \leftarrow$  la taille de  $E$ 
3:  $S \leftarrow$  Une matrice de taille  $T*T$ 
4: for  $I$  allant de 0 à  $T$  do
5:   for  $J$  allant de 0 à  $T$  do
6:      $V = S[i] + D[i]$ 
7:      $V[i] = -\infty$ 
```

```
8:       $V[k] = -\infty$ 
9:       $R[i, k] = R[i, k] + (S[i, k] - \max(v))$ 
10:     end for
11: end for
```

B.4 Méthode de calcul de la disponibilité

```
1:  $E \leftarrow$  L'ensemble d'éléments
2:  $T \leftarrow$  la taille de E
3:  $R \leftarrow$  Une matrice de taille T*T
4: for I allant de 0 à T do
5:   for J allant de 0 à T do
6:      $V \leftarrow$  prendre colonne k dans R
7:     if  $i \neq k$  then
8:        $V[i] = -\infty$ 
9:        $V[k] = -\infty$ 
10:       $V[V < 0] = 0$ 
11:       $D[i, k] = D[i, k] + \min(0, R[k, k] + somme(V))$ 
12:    else
13:       $V[k] = -\infty$ 
14:       $V[V < 0] = 0$ 
15:       $D[k, k] = D[k, k] + somme(V)$ 
16:    end if
17:  end for
18: end for
```

References

- [1] Y. Hirate and H. Yamana. Generalized sequential pattern mining with item intervals. *J. Comput.*, 1(3):51–60, 2006.
- [2] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604, 2007.