

# POM

## Analyse de l'activité et de l'origine des fonds des *exchanges*

Abdullatief Noauffal  
El Azouzi Mohammed

Juin 2022

Résumé : ce projet de recherche a pour but d'analyser et de découvrir l'origine des fonds des *exchanges* de cryptomonnaie, c'est-à-dire leur source initial de fonds afin de réaliser des transactions. Pour cela, nous avons à disposition environ 230 gigaoctets de données provenant de WalletExplorer [1].

## 1 Introduction

Dans le cadre du projet d'orientation master, nous devons travailler sur un projet durant un semestre. Nous avons choisi un projet qui s'inscrit dans le thème *data*, en vue d'intégrer un master 2 Data Science. Nous avons mené ce projet en binôme étant donné que nous avons les mêmes objectifs sur le court terme. Ce projet rallie cryptomonnaie et analyse de données qui sont deux concepts qui nous intéressent particulièrement. Notre projet a été encadré par Rémy Cazabet, membre et chef adjoint de l'équipe Data Mining and Machine Learning (DM2L), au laboratoire du LIRIS.

## 2 Phase d'analyse

### 2.1 Contexte

Tout d'abord nous avons le processeur AMD Ryzen 3 2200G, avec 16 Go de RAM ainsi que de l'espace mémoire externe (environ 250 Go SSD). Nous avons à disposition 230 Go de données, elles représentent les transactions des acteurs de janvier 2009 à janvier 2021. L'objectif étant de trouver l'origine des fonds d'un acteur, il nous a fallu mettre en place un plan. Premièrement, nous avons divisé ce problème en deux problèmes de rang 2 (i.e. sous-problème) : trouver un acteur intéressant sur qui mener nos recherches et restreindre les données sur une période plus courte afin de faciliter les potentiels calculs. Nous avons réduit le premier problème de rang 2 au problème de rang 3 suivant : Analyser les différents acteurs et leur capital, ainsi nous pourrions comparer les acteurs entre eux afin de trouver un candidat potentiel. Le deuxième sous problème de rang 2 a été réduit au problème de rang 3 suivant : Analyser les transactions au cours du temps, en résolvant ce problème, nous pourrions restreindre les données à une période intéressante pour mener nos recherches.

Résoudre les problèmes de rang 3 permettra de résoudre les problèmes de rang 2 qui eux-mêmes apporteront une réponse au problème initial.

### 2.2 Outils d'analyse

Pour ce projet, nous avons décidé de programmer en python car ce langage est simple d'utilisation, il met à disposition une grande variété de bibliothèque intéressante pour l'analyse et le traitement de données, de plus python possède une grande communauté qui pourra nous être très utile durant nos recherches.

#### 2.2.1 WalletExplorer

WalletExplorer [1] est un site web répertoriant des millions de transactions de cryptomonnaies. C'est de ce site que proviennent nos données. Nous avons accès aux transactions des *exchanges* tel que Binance, des services ou même des *pools* de minage.

## Wallet Huobi.com [\(link to service, show wallet addresses\)](#)

Other wallets: | current | 2 |

Page 1 / 1611 [Next...](#) [Last](#) (total transactions: 161,072)

[Download as CSV](#)





date		received/sent	balance	transaction
2021-01-31 14:11:30	 <a href="#">[4006e2c893]</a>	+0.005	17.01218389	<a href="#">b4cf5767a5107e98b11d...</a>
2018-11-09 11:49:28	 <a href="#">CoinJoinMess</a>	+0.0054566	17.00718389	<a href="#">f0d8ae56ef0b8c0ea430...</a>
2018-03-11 11:24:30	 <a href="#">CoinJoinMess</a>	+0.00524327	17.00172729	<a href="#">5d500b55290e8a01d597...</a>

FIGURE 1 – WalletExplorer : Porte-feuille de Huobi.com

Sur la figure 1, nous avons 3 entrées/sorties du porte-feuille de Huobi.com. Une transaction (une ligne), peut agréger plusieurs transactions, la somme de ces transactions est indiquée dans la colonne « *received/sent* ». Nous allons maintenant nous intéresser à une transaction en détails.

### Transaction **b4cf5767a5107e98b11d**<sub>efda47e9c5135a7b7e925863b0c4c8b8b4d6dae0649</sub>

<b>Txid</b>	b4cf5767a5107e98b11defda47e9c5135a7b7e925863b0c4c8b8b4d6dae0649
<b>Included in block</b>	668487 (pos 1562)
<b>Time</b>	2021-01-31 14:11:30
<b>Sender</b>	 <a href="#">[4006e2c893]</a>
<b>Fee</b>	0.00001589 BTC (7.03 satoshis/byte)
<b>Size</b>	226 bytes



inputs: 1 (3.00165582 BTC)	outputs: 2 (3.00163993 BTC)	unique addresses: 2, spent: 1
o. <a href="#">15b8dtbFqeQqJQXXTS1NcNeohYWVD8dKLw</a> 3.00165582 BTC $\Leftarrow$ <a href="#">27d31423...</a>	o. <a href="#">1MTKTCg7Xh3yinQ87sjUzp73ES5caET61t</a>  <a href="#">Huobi.com</a> 0.005 BTC unspent 1. <a href="#">1MmyM89xbEAEqWVs6ga7b9QAMuU5P7CNnd</a>  <a href="#">[de85474010]</a> 2.99663993 BTC <a href="#">731322f5...</a>	

FIGURE 2 – WalletExplorer : Transaction de 4006e2c893 vers Huobi.com

Sur la figure 2, nous avons une transaction de l'acteur 4006e2c893 de 3.00163993 BTC (bitcoin) vers deux adresses, 0.005 BTC vers Huobi.com, vu dans la figure 1 et 2.99663993 vers une autre adresse (de85474010). On constate qu'il y a deux adresses dans la colonne *inputs*, la première adresse est une des adresses du portefeuille du « *sender* », la seconde adresse de la colonne *input* (toujours sur la même ligne), est l'adresse depuis laquelle il a reçu 3.00165582 BTC. Ce sont une partie de ces BTC là que 4006e2c893 a retransmis vers les deux *outputs*. Il y a également deux adresses sur la deuxième ligne de la colonne *outputs*, cela signifie donc que de85474010 a retransmis une partie des BTC qu'il a reçu vers la seconde adresse.

On remarque que nous avons beaucoup d'informations sur ces figures. Ce sont les données avec lesquelles, nous allons travailler pour ce projet.

### 2.2.2 PySpark

Comme nous l'avons dit plus tôt, nous avons choisi de travailler avec python pour ses nombreuses bibliothèques, PySpark en fait partie. PySpark va nous permettre de résoudre un des problèmes annexes de ce sujet : la manipulation de 230 Go de données. Lorsque l'on parle de traitement de données sur python, on pense immédiatement à la bibliothèque pandas. Cependant, lorsqu'on a affaire à des données trop massives, les calculs deviennent trop lents. Heureusement, il existe une autre bibliothèque python, assez proche de pandas, qui permet de traiter des très grandes quantités de données : PySpark. Grâce à cette librairie, nous pouvons requêter directement les données.

PySpark accélère le traitement de données en utilisant le calcul distribué, technique qui consiste à exploiter plusieurs unités de calcul réparties en clusters au profit d'un seul projet afin de diviser le temps d'exécution d'une requête.

### 2.2.3 Bibliothèque de data visualisation

Matplotlib est une bibliothèque Python capable de produire des graphes de qualité. Matplotlib peut être facilement utilisé dans des scripts. Il va nous être particulièrement utile pour visualiser l'évolution d'une donnée au cours du temps.

NetworkX est une bibliothèque Python pour la création, la manipulation et l'étude de la dynamique et des fonctions de réseaux complexes. Combinée avec Pyvis qui est également une bibliothèque de génération rapide de réseau visuel (graphe), nous pourrions visualiser les potentiels liens entre les données.

## 3 Phase de développement

Après avoir analysé les données en détail, pris en main les outils, nous avons mis en exécution notre plan.

### 3.1 Période

Pour restreindre les datas, nous devons d'abord analyser les transactions. Nous avons donc commencé par calculer le nombre de transactions par année et le nombre de transactions cumulées par année.

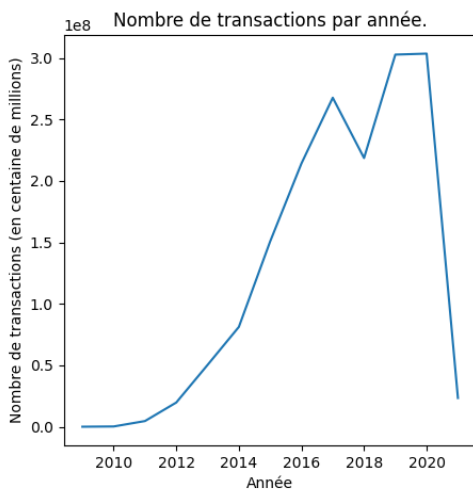


FIGURE 3 – transactions par année.

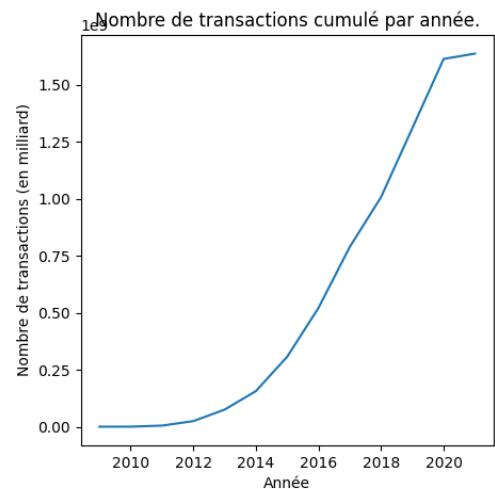


FIGURE 4 – transactions cumulée par année.

D'après nos analyses, il y a eu environ 150 millions de transactions en 2015 avec plus de 300 millions de transactions cumulées jusqu'en 2016, contre plus de 300 millions de transactions en 2020 avec plus de 1.5 milliards de transactions cumulées jusqu'en 2021. Nous avons décidé avec l'aide de notre encadrant qu'il était plus judicieux de travailler sur les données sur la période de 2009 à 2016 (exclus) pour nos calculs car on divise les données par un facteur 5, on passe de 1.5 milliards de transactions à 300 millions. Cette décision ne faussera pas nos calculs car les 300 millions de transactions restantes sont largement suffisantes pour mener à bien nos recherches.

### 3.2 Acteurs

À présent, nous pouvons nous concentrer sur le choix des acteurs que nous allons étudier. Pour cela, il a fallu commencer par analyser les acteurs.

#### 3.2.1 Volume encaissé

Premièrement, nous avons requêté les données pour obtenir les acteurs qui ont le plus encaissé depuis leurs créations jusqu'en 2016. Cette simple requête nous a pris plus de 6 minutes sur notre machine (la machine décrite

dans le contexte), elle prend environ 4 fois plus de temps sur la machine personnelle du binôme. Les résultats nous montrent seulement les acteurs (de type *exchange*) ayant le plus gros volume encaissé. Parmi cela se trouve Epay.info en première position, Huobi.com-2 en deuxième position et enfin Bitsamp.net en troisième position. Mais cela ne suffit pas pour choisir un acteur sur lequel travailler, en effet il se peut qu'un acteur ait reçu 1 million de dollars (USD) et qu'il l'ait par la suite retransmis à un autre acteur. Dans ce cas, son volume encaissé s'élève à 1 million USD alors que son capital net est à 0.

### 3.2.2 Capital net

Il nous a fallu calculer le capital net des acteurs et choisir quelques acteurs parmi ceux qui ont un capital net des plus élevés avant 2016. Après réflexion, nous ne pouvions calculer le capital net de chaque acteur en une seule requête. Nous l'avons donc calculé en 2 requêtes, la première étant le volume encaissé par acteur, la deuxième requête est le volume dépensé par acteur. Il fallait sauvegarder les résultats dans des dataframes différents pour ensuite faire la différence des sommes pour obtenir le capital net par acteur. Mais il y a un problème avec cette méthode : lors de la requête, le programme se termine avec une erreur car il n'y a pas assez de mémoire pour stocker ces données dans une variable.

### 3.2.3 Les recherches

Nous avons combiné les résultats obtenus pour le volume encaissé à des recherches sur internet et voici ce que nous avons trouvé : En 2016 Huobi.com fait partie du top 3 des plus gros *exchanges* et Bitstamp.net ainsi que Huobi.com-2 sont dans le top 15 des *exchanges* aujourd'hui selon CoinMarketCap [2].

### 3.2.4 Analyse du capital de Huobi.com-2 et de Bitstamp.net

Pour confirmer nos choix d'acteurs, nous avons analysé l'évolution du capital de Huobi.com-2 et de Bitstamp.net afin d'être sûr que les données sont cohérentes.

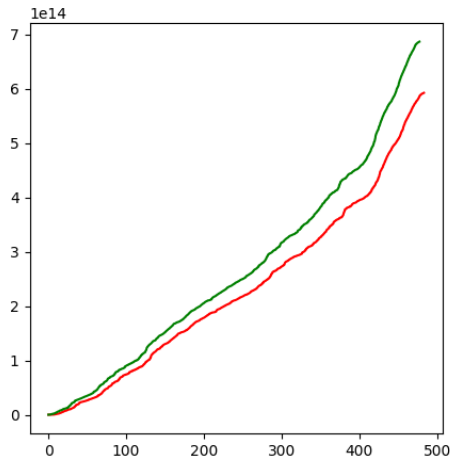


FIGURE 5 – Evolution des gains et des dépenses de Huobi.com-2 jusqu'en 2016.

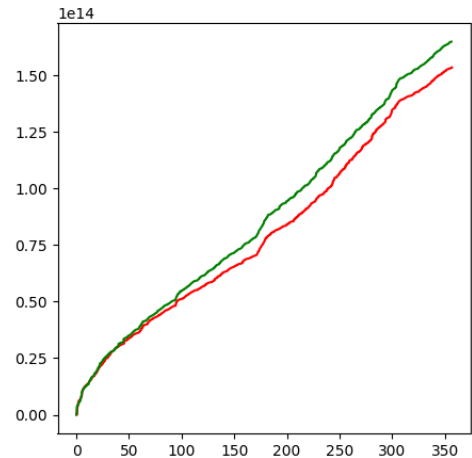


FIGURE 6 – Evolution des gains et des dépenses de Bitstamp.net jusqu'en 2016.

Les figures 5 et 6 représentent l'évolution des gains (courbe verte) et dépenses (courbe rouge) de chaque acteur en satoshi. L'évolution ne se fait pas sur la même durée car les transactions de Huobi.com-2 ont commencé à environ 500 jours avant le 31/12/2015 tandis que les transactions de Bitstamp.net ont commencé seulement 350 jours avant.

Nous avons donc décidé de travailler avec Huobi.com-2 et Bitstamp.net. Nous avons mis Epay.info de côté car notre encadrant nous a prévenus que les données sur Epay.info étaient étranges et qu'il valait mieux l'éviter.

### 3.3 Recherche de l'origine des fonds d'un ou plusieurs acteurs

Dès lors, nous pouvions nous attaquer au coeur du sujet, l'objectif est toujours de découvrir l'origine des fonds de Huobi.com-2 et de Bitstamp.net. Y en a-t-il plusieurs? Un seul? Sont-ils connus? Sont-ils des *exchanges* ou bien des acteurs inconnus?

#### 3.3.1 Algorithme A

Cet algorithme consiste dans un premier temps à analyser le premier mois d'activité de chaque acteur passé en paramètre. Pour chaque acteur passé en paramètre, l'algorithme calcul et stock dans un dictionnaire le montant total net reçu des autres acteurs et retourne une liste triée des acteurs (avec le montant échangé) qui ont le plus gros volume échangé en net, puis on réitère l'algorithme sur les acteurs ayant les plus gros montant envoyé aux acteurs passés en paramètre. L'algorithme va analyser leurs premiers mois et ainsi de suite jusqu'au rang n.

##### a) Volume encaissé de chaque acteur

Dans la première étape, l'algorithme calcule le montant envoyé à Bitstamp.net (resp. Huobi.com-2) depuis chaque acteur. Pour cela, on requête les données pour récupérer uniquement les données utiles ensuite nous les stockons dans un dataframe pour effectuer nos calculs. Voici un exemple de résultat une fois stocké dans un dictionnaire.

```
{ 'Bitstamp.net': { '0': 1465.279, '10009111': 3809.838, '10019759': 1.682 },  
  'Huobi.com-2': { 'Coin.net': 548.36, '54842541': 3331.23, '111023': 0.39 } }
```

Dans ce dictionnaire Bitstamp.net a reçu au total 1465.279 USD de l'acteur '0' durant son premier mois.

##### b) Volume envoyé à chaque acteur

Dans la seconde étape, l'algorithme calcule le montant total envoyé à chaque acteur durant le premier mois et le stock dans un nouveau dictionnaire semblable au premier.

```
{ 'Bitstamp.net': { '0': 1200.152, '10009111': 452.32, '10019759': 1.44 },  
  'Huobi.com-2': { 'Coin.net': 563.36, '54842541': 444.2553, '111023': 35.39 } }
```

Dans ce dictionnaire Huobi.com-2 a envoyé au total 563.36 USD à l'acteur Coin.net durant son premier mois.

##### c) Montant net échangé avec chaque acteur (entrée ou sortie)

Dans un troisième temps, l'algorithme calcule la différence entre le montant qu'il a reçu de chaque acteur et le montant qu'il leur a envoyé pour obtenir le montant net échangé et stock le résultat dans un dictionnaire final.

```
{ 'Bitstamp.net': { '0': 265.127, '10009111': 3,357.518, '10019759': 0.242 },  
  'Huobi.com-2': { 'Coin.net': -15, '54842541': 2,886.9747, '111023': -35 } }
```

Ce dictionnaire représente les montants nets que Huobi.com-2 et Bitstamp.net ont reçu des autres acteurs. Huobi.com-2 a reçu 2886,9747 USD de l'acteur '54842541' et a envoyé 15 USD à l'acteur Coin.net. Ce dictionnaire est trié par ordre décroissant du montant avant d'être retourné.

##### d) Visualisation et récursivité

Enfin, nous avons modélisé les résultats sous forme de graphe. Comme nous l'avons montré précédemment, le dictionnaire est du format :

```
{ 'acteur passe en parametre': { 'acteur voisin': montant, 'acteur voisin': montant } }
```

Ce format nous permet de créer un graphe de ce type facilement :

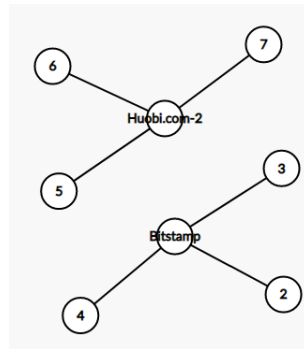


FIGURE 7 – Exemple de graphe de rang 1

Dans le graphe, les nœuds 5, 6 et 7 sont les acteurs qui ont envoyé le plus gros montant à l'acteur Huobi.com-2, ils représentent le rang 1. L'objectif à présent était d'étendre le graphe à des rangs plus élevés, c'est-à-dire obtenir les trois acteurs qui ont envoyé le plus gros montant aux acteurs 5, 6 et 7. Pour cela, nous avons eu recours à la récursivité. La création du graphe se fait donc au fur et à mesure de la récursivité. À chaque rang, on lance l'algorithme A, on récupère le dictionnaire final issu de l'algorithme. On ajoute au graphe les clés du dictionnaire (Bitstamps.net et Huobi.com-2 pour le rang 1 c.f. figure 7) et ensuite pour chaque clé, on ajoute au graphe les 3 premiers acteurs des dictionnaires correspondant à chaque clé avec un lien entre l'acteur et la clé. On passe ensuite à la boucle suivante, l'algorithme A est relancé cette fois-ci sur les acteurs que l'on vient d'ajouter au graphe en excluant les clés. (2, 3, 4, 5, 6, 7).

### 3.3.2 Algorithme B

Le problème de l'algorithme A est que pour chaque acteur de départ, il fait ses calculs sur l'intervalle [date de la première transaction de l'acteur de départ ; date de la première transaction de l'acteur de départ + 1 mois] (cf. Annexe 1), et ce pour chaque rang. Or nous nous intéressons à l'origine des fonds de chaque acteur étudié lorsque l'on remonte les rangs, pour enfin savoir l'origine de tous les fonds et si cette dernière peut être simplement catégorisée.

Nous ne pouvions pas gérer tous les acteurs du même rang en une étape car nous devons gérer différents intervalles de temps pour chaque source de chaque acteur. Pour l'intervalle, on choisit du début de l'acteur traité jusqu'à un mois après le début de l'acteur du rang précédent pour lequel l'acteur traité est donneur (cf. Annexe 2). Nous avons donc choisi de faire une fonction qui récupère pour un acteur tous ses donneurs sur une période – en prenant pour base l'algorithme A –, puis nous avons mis l'appel à la fonction dans une boucle pour chaque acteur. La boucle va récursivement traverser chaque rang et prendre les 3 plus gros donneurs nets et la somme nette donnée par les autres acteurs. Tous ces choix ont été faits dans un souci de performances, et afin de ne pas traiter des données inutiles. Le format des données récupéré est le même que pour l'algorithme A.

## 3.4 Optimisation

L'exécution de ces algorithmes dure au minimum 1h30 (algorithme A jusqu'au rang 3). Pour réduire ce temps, il a fallu optimiser les algorithmes.

### 3.4.1 Pré traitement des données

Pour que les algorithmes fonctionnent, il doit connaître la date de la première transaction de chaque acteur. Pour un seul acteur, il serait simple de récupérer le *time* de la première transaction, ce serait tout simplement le plus petit *timestamp*. Mais lorsqu'il y en a plusieurs, il faut associer à chaque acteur le plus petit *timestamp* de ses transactions. Ceci est faisable soit avec une boucle *for* afin de traiter chaque acteur individuellement, soit avec une requête imbriquée. Dans les deux cas le temps était trop élevé, cela est dû au nombre d'acteurs passés en paramètre qui augmente à chaque rang. Nous avons donc créé un nouveau data set avec une donnée en plus : le *begin*. Voici la requête :

```

res =
spark.sql("SELECT src_identity , dst_identity , valueUSD , year , month ,
              day , time , begin ,src_cat ,dst_cat " +
          "FROM data NATURAL JOIN (SELECT DISTINCT dst_identity , MIN(time) AS begin " +
          "FROM data " +
          "WHERE year<=2015 GROUP BY dst_identity) " +
          "WHERE year<=2015 and dst_identity <> src_identity")
res.createOrReplaceTempView('data_temp')

```

chaque ligne de la table *data\_temp* représente une transaction simplifiée avec le strict nécessaire aux calculs. Grâce à ce pré traitement, nous avons réduit le temps d'exécution d'environ 3 pour l'algorithme A.

### 3.4.2 Localité des données

Une autre optimisation que nous avons faite est le choix de la localité des données. initialement les données ont été placées sur un disque dur externe HDD ce qui n'était pas optimal. Nous avons ensuite mis la main sur un disque dur externe SSD et avons téléversé les données du disque HDD vers le disque SSD. Nous avons eu de meilleures performances grâce à cette optimisation cela est notamment dû à la vitesse de lecture du disque SSD qui est largement supérieur à celui du HDD.

## 4 Résultats

### 4.1 Résultats de l'algorithme A

#### 4.1.1 Étude

Une fois l'algorithme A exécuté, le graphe est généré dans un fichier HTML à l'aide de Pyvis. Voici le graphe de Bitstamp.net :

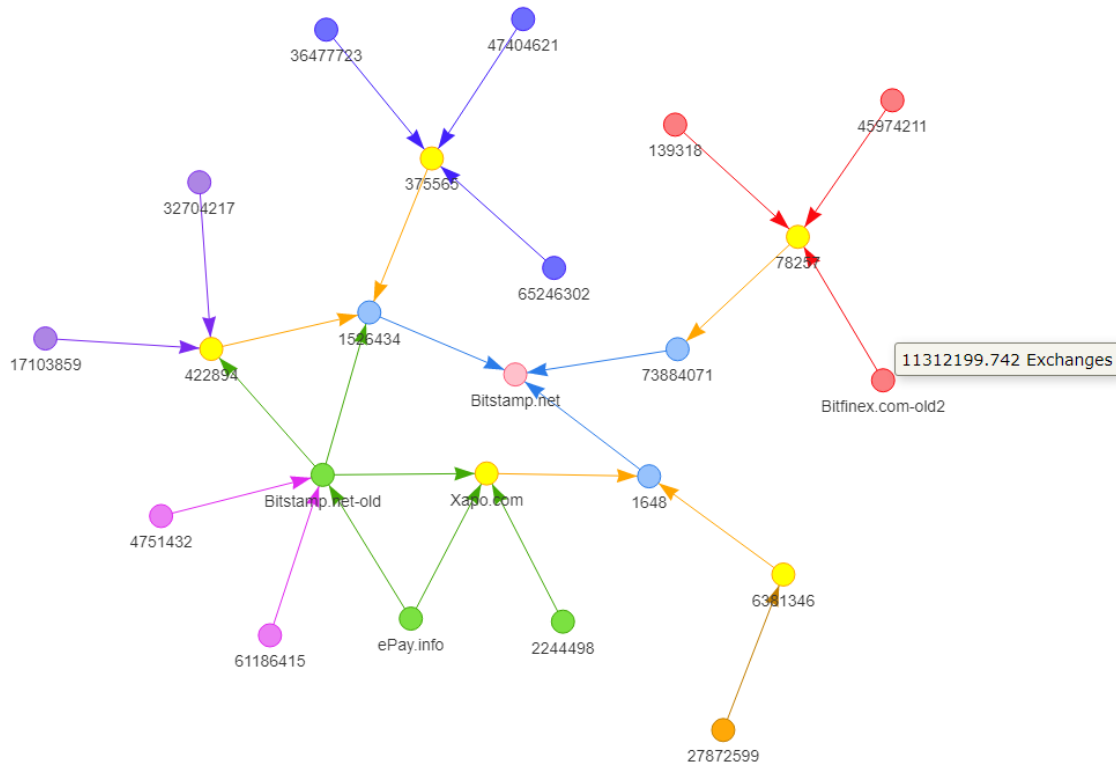


FIGURE 8 – Résultat Bitstamp rang = 4 algo A.

Chaque noeud représente un acteur, le centre de ce graphe de rang 3 est Bitstamp.net. Les liens représentent le montant net envoyé à un acteur. Chaque acteur reçoit de trois autres acteurs, si un acteur reçoit d'un nombre d'acteurs inférieur à 3, cela signifie que durant son premier mois, l'acteur en question a échangé avec un ou deux acteurs. En survolant un noeud à l'extrémité du graphe, nous pouvons voir le montant envoyé au noeud au quelle il est relié ainsi que sa catégorie et en survolant un noeud qui n'est pas à l'extrémité nous pouvons voir la somme des montants reçu des trois autres acteurs.

#### 4.1.2 Interprétation

D'après nos résultats, on voit que Bitstamp.net perçoit 17 Millions de USD durant son premier mois de la part de 3 acteurs qui eux-mêmes reçoivent durant leurs premiers mois des montants d'autres *exchanges* comme Bitstamp.net-old et Xapo.com. Bitstamp.net-old est un autre prote-feuille de Bitstamp.net, nous pouvons donc supposer que l'acteur 1526434 est un intermédiaire entre Bitstamp.net-old et Bitstamp.net. On peut aussi supposer que Xapo.com prête ou donne un certain montant à Bitstamp.net via un intermédiaire qui serait l'acteur 1646. Si on remonte jusqu'aux extrémités du graphe, la majorité des noeuds ont une catégorie *None*, ce qui signifie qu'on ne les connaît pas. De plus, nous pouvons n'émettre que des suppositions car il est possible que durant le premier mois des autres acteurs, Bitstamp.net n'existe pas. Avec ce résultat, nous pouvons seulement avoir une idée très globale des échanges. En augmentant le rang, il serait peut-être possible de tomber sur des *pools* de minage qui pourrait être une source potentielle d'où pourraient provenir les fonds de bitstamps.net.

#### 4.1.3 Limites

Ces résultats sont très limités car nous avons une vue d'ensemble, ce qui n'est pas précis. On analyse seulement le premier mois de chacun donc il y a une perte d'informations importantes. De plus le rang étant de 3, nous ne pouvons pas affirmer qu'un acteur soit la source de Bitstamp.net, il faudrait un rang plus élevé ce qui prendrait beaucoup de temps et même dans ce cas, nous ne pourrions rien affirmer puisque nous analysons seulement le premier mois.

**Ce graphe reste tout de même très intéressant car il peut en évidence certains acteurs (source potentielle des fonds) qui fournirait des gros montants à plusieurs acteurs(durant leur premier mois). C'est dans cet optique qu'a été pensé cet algorithme.**

## 4.2 Résultats de l'algorithme B

### 4.2.1 Étude

Après une longue exécution de plusieurs dizaines d'heures (environ 30 heures), l'algorithme génère un graphe comme pour l'algorithme A et il génère aussi un diagramme Sankey. Ce diagramme permet la visualisation de flux entre les différents rangs. L'épaisseur des liens correspond au volume monétaire net donné. En figure 9 et 10 ci-après, les diagrammes pour Bitstamp.net comme acteur de départ.

Sur le dessin Sankey, on distingue chaque rang par les noms des acteurs (se terminant par -rX) et par l'alignement vertical des acteurs apparaissant à ce rang. Les acteurs peuvent intervenir à plusieurs rangs. Le libellé "autre" correspond à la somme des autres donneurs puisque l'on prend uniquement. Le graphique a la même forme qu'avec l'algorithme A.

### 4.2.2 Interprétation

Les résultats du graphique sont similaires à l'algorithme A, on n'observe pas de singularité particulière, on voit qu'il y a des liens et des acteurs en plus. C'est normal puisqu'on changes les intervalles. On voit qu'il y a d'autres acteurs en donneurs, mais aussi des acteurs inconnus, la répartition est assez homogène. Sur le Sankey, on observe que plus le rang est élevé, plus les donneurs se croisent. Les acteurs apparaissent à plusieurs rangs. On voit que l'acteur 1648 reçoit énormément de fonds, mais n'en envoie que très peu à Bitstamp.net. 1648 appartient peut-être à Bitstamp, il s'agirait alors d'un intermédiaire. Ou peut-être est-il un gros acteur qui a été donneur de Bitstamp.





FIGURE 9 – Sankey Bitstamp pour une exécution jusqu’au rang 5.

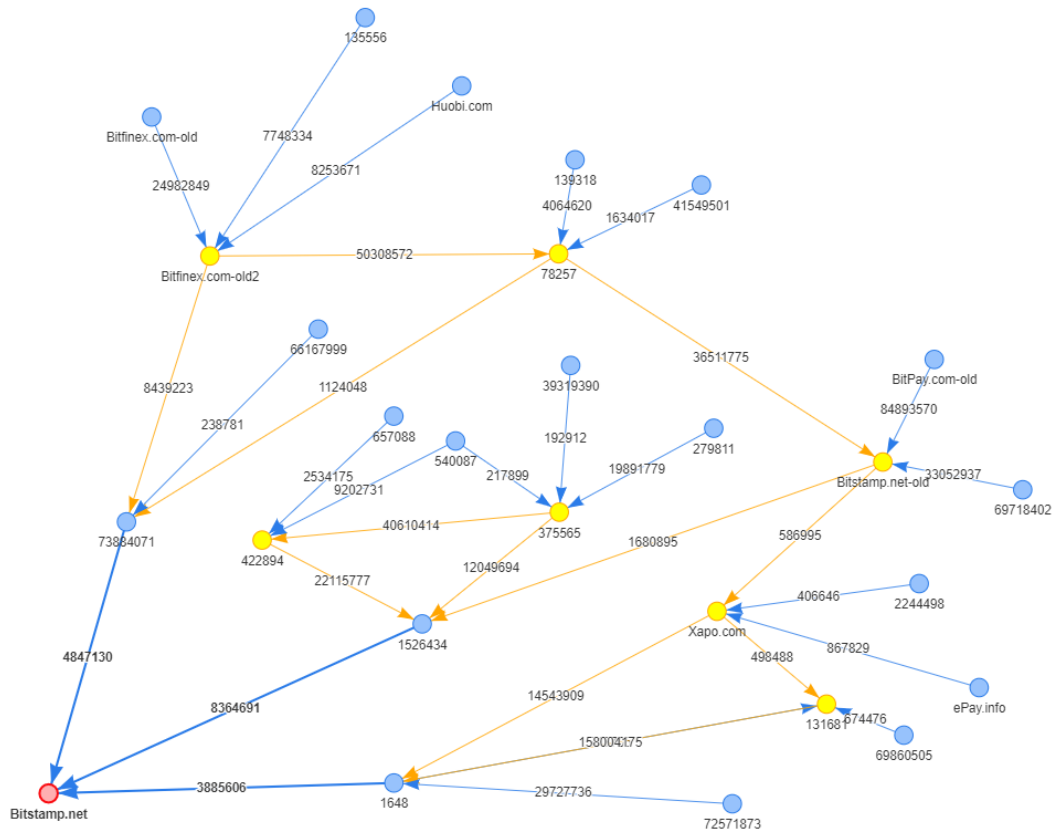


FIGURE 10 – Résultat Bitstamp rang = 4 algo B.

### 4.2.3 Limites

Une limite de cet algorithme B est qu'il ne trace pas exactement la cible des fonds transférés à travers des rangs. Puisque notre jeu de données ne contient pas les sorties multiples des transactions, on ne peut pas résoudre les chaînes de transactions afin d'avoir une provenance et une cible plus précises. En effet, sur la blockchain BTC, une transaction peut avoir plusieurs destinations et déboucher sur d'autres transactions. De plus, notre algorithme ne calcule pas les intervalles en fonction de la somme donnée ; par exemple, on pourrait récupérer la date après laquelle un acteur a reçu l'équivalent de la somme qu'il a donnée à un autre, avant qu'il ne lui donne.

## 5 Problèmes rencontrés

### 5.1 Quantité massive de données

Le premier problème rencontré est la gestion des données. On possède chacun une machine, un ordinateur portable pour l'un et la machine décrite dans le contexte pour l'autre. Nos 2 machines ne pouvaient accueillir autant de données (mémoire restante insuffisante). Notre encadrant nous a donc prêté un disque dur externe avec les données dessus. Nous avons mis à profit ce disque sur la machine la plus puissante. Le binôme avec l'ordinateur portable travail uniquement sur des données très réduites. Par la suite nous avons réussi à récupérer un disque dur externe afin que chacun travail dans de bonnes conditions.

### 5.2 Exécution des calculs trop longue

Comme dit plus haut, nous travaillons sur un jeu de données de taille élevé, beaucoup trop élevée pour le charger en RAM, et Pyspark règle ce problème en traitant les données pendant qu'il les lit. Le problème est qu'il est bien plus long d'accéder à des données sur un disque dur que sur de la RAM. Se pose donc un soucis quant au temps des requêtes et à leur intégrité avant l'exécution. Il n'y a pas vraiment de solution miracle pour pallier à cela. Nous avons premièrement optimisé nos requêtes en restreignant les intervalles sur lesquelles elles s'exécutent, puis nous avons préenregistré des jeux de données traités qui nous semblaient pertinents et cela nous a permis d'accélérer grandement nos exécutions, passant de quelques heures à quelques minutes. Enfin, il a également fallu que nous soyons méticuleux pour bien vérifier avant chaque exécution les requêtes.

### 5.3 Précision des données retenues

Les données retenues pour notre travail sont un jeu de données déjà traité en amont pour simplifier notre tâche, mais cela le rend moins précis. Pour remonter à l'origine des fonds, il aurait été pertinent de partir des données brutes, mais comme montré dans les résultats, cela n'aurait pas changé la conclusion, donc c'est un problème seulement pour la précision des résultats. Nous n'avons donc rien changé à ce niveau-là.

## 6 Conclusion

En guise de conclusion, on peut rappeler les résultats ; ils ne montre vraisemblablement aucun indice de source de fonds commune à tous les acteurs, ni même une source unique pour chaque acteur étudié. Les résultats montrent qu'il y a entremêlements et enchevêtrements de liens entre les différents acteurs et que même en remontant, on arrive à la conclusion que les acteurs et leurs donneurs échangent entre eux et avec eux-mêmes, qu'ils échangent avec d'autres *exchanges*, avec des mineurs et avec des acteurs inconnus... Cependant, ces résultats sont à nuancer car les données utilisées n'ont pas été les plus précises et car il faudrait également enquêter en détail sur les *exchanges* eux-mêmes, en plus de leurs transactions sur le réseau BTC. La réponse à la question que nous nous sommes posée n'est donc pas tranchée, il semblerait que la source des fonds des *exchanges* soit d'origine hétérogène.

## 7 Annexes

### 7.1 Annexe 1

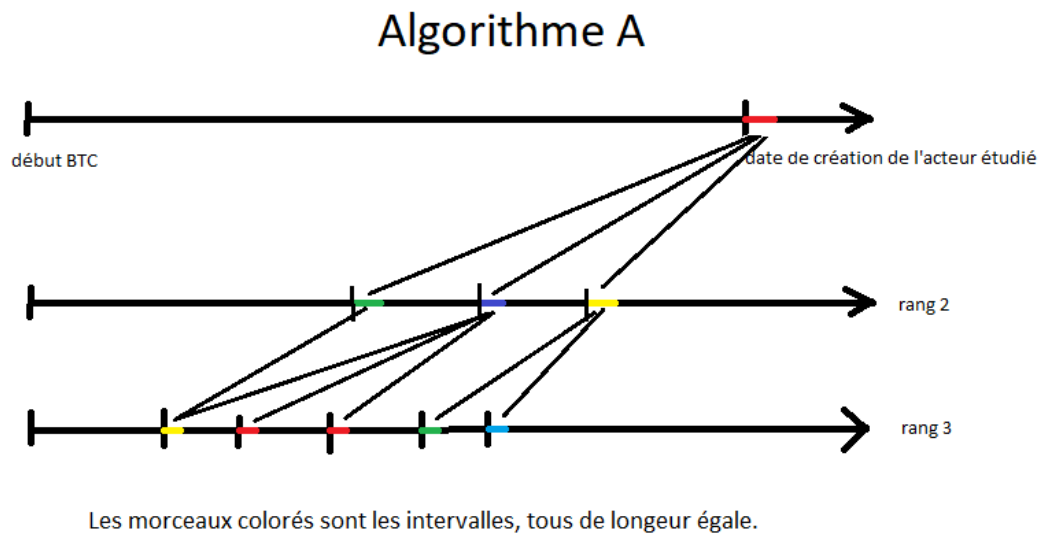


FIGURE 11 – intervalle de temps (1mois) algorithme A.

### 7.2 Annexe 2

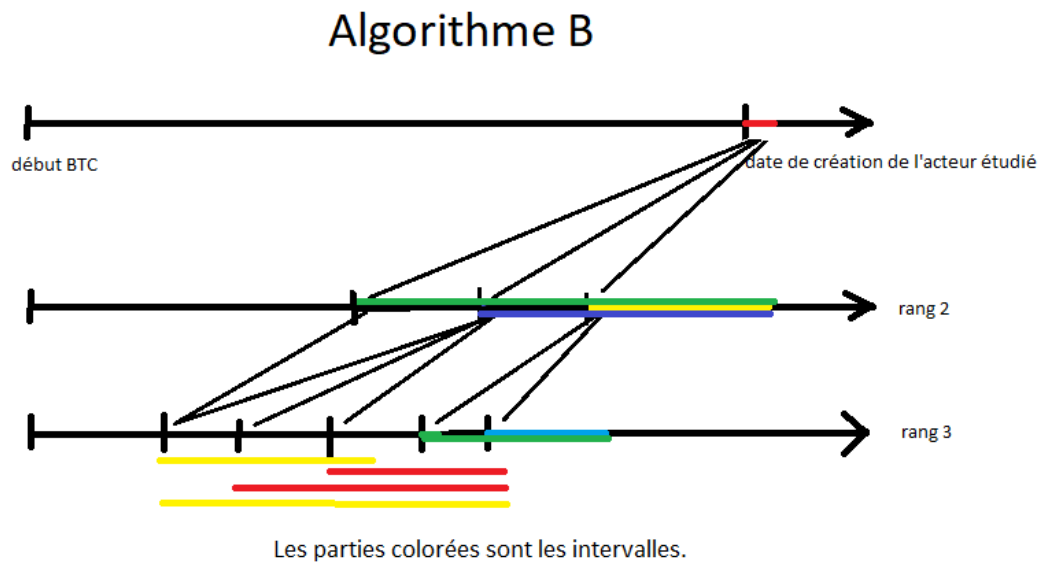


FIGURE 12 – intervalle de temps (variable en fonction des acteurs) algorithme B.

## Références

- [1] <https://www.walletexplorer.com/>
- [2] Brandon C. (May 2013) – <https://coinmarketcap.com/fr/rankings/exchanges/>