Bolong ZHANG
Fangda ZHU
Yukun BAO
Corentin ROBINEAU

Projet IGR205: Interactions sur des graphes de connaissances interconnectés

Github: https://github.com/BolongZHANG/IGR205_Graphes

Déroulement des recherches

Nous avons tout d'abord effectué des recherches afin de trouver des articles intéressants sur les différentes manières d'explorer les graphes RDF. Au début, nous souhaitions essayer d'explorer le graphe tout entier afin de permettre aux utilisateurs de pouvoir découvrir les classes et les prédicats les plus importants pour ensuite pouvoir faire des requêtes dépendantes de la structure du graphe. En effet, dans beaucoup de projets et de papiers, les algorithmes reposent sur des requêtes qui sont basées sur la connaissance de la structure du graphe. Cependant, nous souhaitions créer un algorithme universel qui puissent découvrir la structure du graphe. Ainsi, nous avons commencé à nous pencher sur les techniques permettant de faire des requêtes qui renvoient les éléments du graphes correspondant à une demande d'un utilisateur. Nous avons décidé que cette demande se ferait sous la forme de mots clefs.

Après plusieurs recherches, nous avons porté notre attention sur l'article suivant: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4812421&tag=1

Cet article propose plusieurs solutions qui permettent de passer d'un ensemble de mots clés à un ensemble des top-k requêtes qui permettent d'obtenir les meilleurs résultats pour ces mots clés sur un graphe donné. Les étapes principales sont les suivantes:

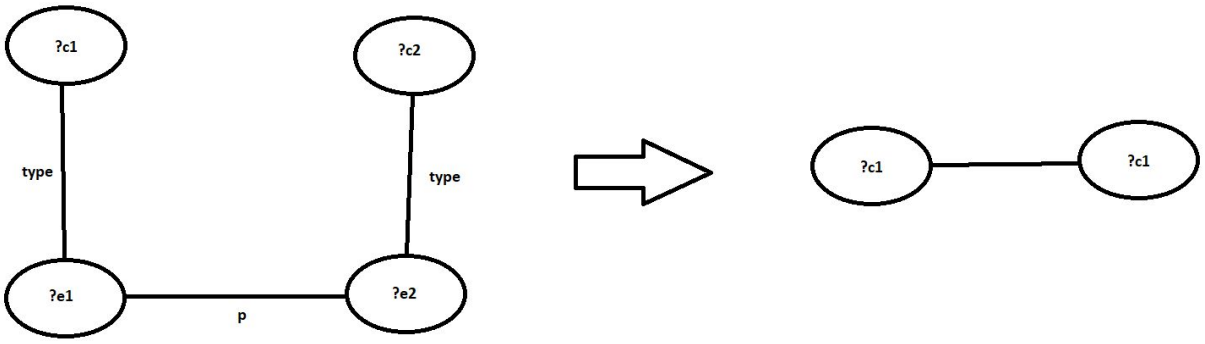
1) Le mapping des mots clefs aux éléments du graphes RDF

C'est à dire que pour chaque noeud de type label qui match avec les keyword on souhaite récupérer toutes les classes auxquelles il est relié.

2) la création d'un graphe "résumé" du graphe d'origine

Le graphe "résumé" est un graphe constitué seulement des sommets de type classe du graphe d'origine. Les arêtes de ce graphe sont les arêtes (prédicats) du graphes d'origine qui relient des entités. Un projet appartenant à la classe Project est une entité. En effet, ce

n’est pas un attribut / donnée car ce projet est composé d’un nom, d’une date de publication,etc. Ces derniers en revanche sont des données.



Une fois le graphe “résumé” obtenu, nous allons créer une version augmentée de celui-ci. En effet, nous allons lui ajouter les noeuds du graphes d'origine qui contiennent les keywords et qui sont reliés aux noeuds du graphe “résumé”. Pour réaliser cela il nous faut utiliser le mapping réalisé précédemment.

3) L’exploration de ce graphe pour trouver des sous graphes connectant les mots clefs
4) Détermination des top-k graphes (les meilleurs) par l'intermédiaire d’une fonction d’évaluation
5) Génération des requêtes SPARQL pour les top-k graphes

Cependant, nous avons rencontré un problème majeure très rapidement. En effet, le graphe résumé indispensable à la réalisation de l’algorithme s’est avéré être irréalisable en pratique car beaucoup trop gros dans la plupart des cas (sauf pour le graphe Sembib). Nous avons donc dû adapter le papier afin de pouvoir récupérer des résultats pour ensuite les visualiser.

Algorithme

Nous sommes intéressés par la question: Comment explorer un large graphe RDF efficacement et intuitivement. SPARQL est un langage prédominant pour réaliser des requêtes sur les graphes RDF. Cependant, il demande à l’utilisateur d'avoir une connaissance parfaite de la syntaxe mais aussi de la structure du graphe.

Avec l’algorithme suivant , nous voudrions combiner la recherche par mots clefs avec le langage de requête SPARQL. Etant donnée une requête (Q , q) où Q est une requête SPARQL et q et un ensemble de mots clés, nous supposons que la force de relation de la réponse dépend de la longueur du chemin . En fait, des prédicats différents devraient avoir des poids différents pour la force de relation et cela peut être mesuré par une métrique de distance. Un autre challenge pour résoudre ce problème et l'efficacité de la recherche. Une recherche exhaustive peut se dérouler de la façon suivante: On cherche tous les sous

graphes qui match la requête Q puis on calcul le plus court chemin entre ces sous graphes et les sommets qui contiennent les keyword. Evidemment, c'est une solution inefficace en pratique. On peut trouver les match de la requête Q avec des heuristiques afin de stopper le processus de recherche le plus tôt possible. De plus, on peut utiliser un schéma de reconnaissance séquentiel afin d'élaguer les branches de recherches. Enfin, pour accélérer le calcul de la distance, on peut sélectionner des pivots et les considérer comme des racines indépendantes dans l'arbre de recherche.

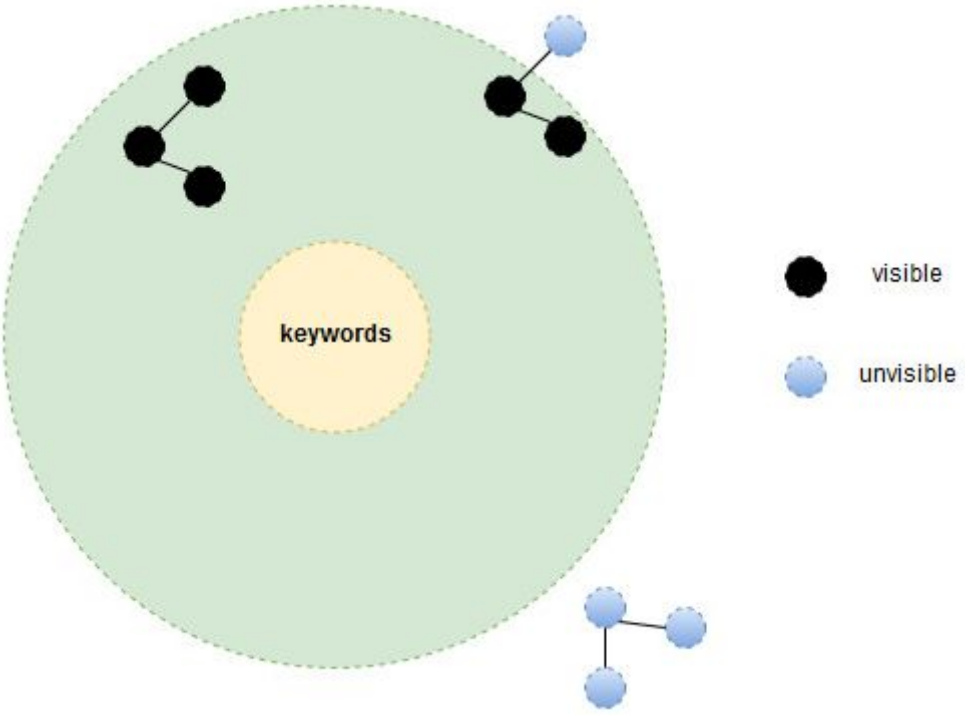
`Jian Pei; Wen Jin\n select * where { ?paper <http://swrc.ontoware.org/ontology#year> "2005" . ?paper <http://swrc.ontoware.org/ontology#booktitle> "@@VLDB" . ?paper <http://purl.org/dc/elements/1.1/creator> ?person2 . }`	Qui a écrit une papier nommé VLDB en 2005 et garde une bonne relation avec Jian Pei et Wen Jin?
`Peking\n select * where { ?p <http://mpii.de/yago/resource /actedIn> ?f1 . ?p <http://mpii.de/yago/resource/created> ?f2 . ?f1 rdf:type <http://mpii.de/yago/resource /wikicategory_Comedy_films> . ?f2 <http://mpii.de/yago/resource /hasProductionLanguage> <http://mpii.de/yago/resource /English_language> . }`	Quels producteurs de films anglais ont joué dans un film de comédie et se sont liés à l'université de Pékin?

On commence par un indexage hors ligne où les données du graphe sont traitées et placées dans des structures de données spécifiques. Du point de vue du graphe, les mots clefs peuvent faire référence à des C-Vertices (les classes), des E-Vertices (les entités) ou des V-vertices (les données) et les arêtes du graphe. Nous ne nous intéresserons pas aux E-vertices dans le processus d'indexage car on peut supposer qu'il est peu probable que les utilisateurs utilisent la verbose des URI d'une E-Vertex. Afin de reconnaître aussi les mots clefs qui ne match pas exactement avec les données, la structure de donnée qui map les éléments du graphe au key word est implémentée sous la forme d'un index inversé. Ainsi, la structure de donnée va analyser un mot clef et retourner une liste d'éléments qui ont des labels avec une sémantique ou une syntaxe similaire.

La partie principale de l'algorithme consiste en trois parties. Tout d'abord, on explore le graphe pour trouver des sommets qui connectent les sommets "mot clef". Ensuite, on génère des match à partir des sommets connectant les sommets "mot clef" et on obtient des centaines de milliers de sous graphes(graphe résumé). Pour finir, on extrait les top-k sous graphes qui correspondent le mieux aux mots clefs de l'utilisateur. Lors de l'exploration du graphe, pour chaque mot clef on a une file de priorité de triplets (sommet , chemin vers un sommet "mot clef" , longueur du chemin). Tous les éléments sont triés dans l'ordre non descendant de la longueur du chemin. Chaque mot clef est aussi associé à un ensemble de résultats dans lequel on peut trouver tous les sommets visibles avec un chemin de longueur inférieur à l'infinie.

Pour la génération des match de sous graphes, on utilise un algorithme basé sur le parcours en profondeur pour réaliser le processus de matching en commençant par un sommet v qui est directement visible par le sommet "mot clef". Supposons que v match un sommet u dans la requête Q, alors on cherche le graphe pour atteindre le voisin de v qui est

un voisin de u et dont l'arête qui les relies est une arête de la requête. Le recherche va s'étendre pas à pas jusqu'à ce que l'on trouve un match ou que l'on ne puisse plus continuer. Pour accélérer la traversée, on sélectionne des sommets que l'on appellera pivots et on calcul les arbres de plus courts chemin ayant pour racine ces pivots. Si la traversée rencontre un pivot, on peut utiliser l'arbre de plus court chemin afin d'explorer l'espace. Enfin, afin de calculer les top-k match, on commence par calculer le coût des match avec les sous graphes pour lesquels les sommets sont directement visibles par les mots clefs, ce qui peut mener à un seuil d'évaluation représentant le k-ième plus petit coût jusqu'à maintenant. Quand il y a moins de k match de ce type, le seuil devrait être infinie. Si le seuil est inférieur à toutes les évaluations des match où tous les sommets ne sont pas visible directement par les mots clefs, l'algorithme s'arretera et renverra les résultats qu'il a trouvé, sinon l'algorithme entame la prochaine itération.



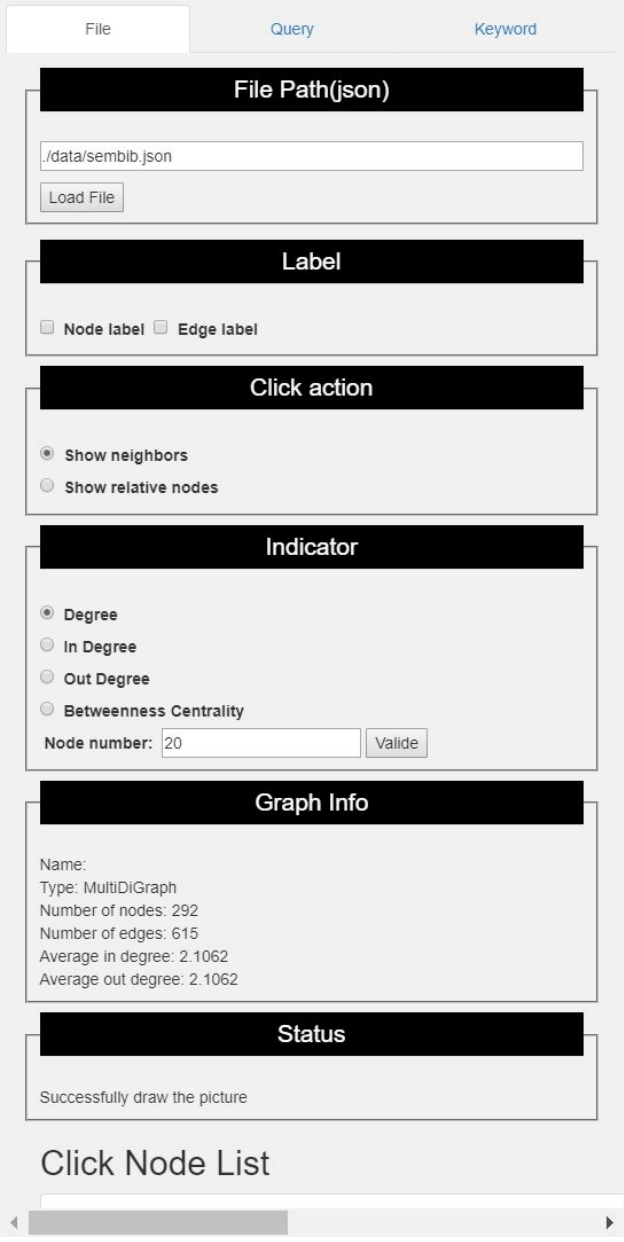
On remarque que les résultats du graphe résumé retournés par l'algorithme sont des triplets et ils peuvent être visualisés comme un graphe. Les réponses à la requête Q ne sont pas nécessairement liées directement aux variables dans la question des utilisateurs. Cependant on peut donner une autre requête de mots clefs et l'algorithme prendra en compte les réponses précédentes ainsi que les nouvelles variables . Ainsi, le deuxième graphe résumé retourné par l'algorithme révèlera de façon intuitive le lien logique entre les réponse et la question.

Visualisation

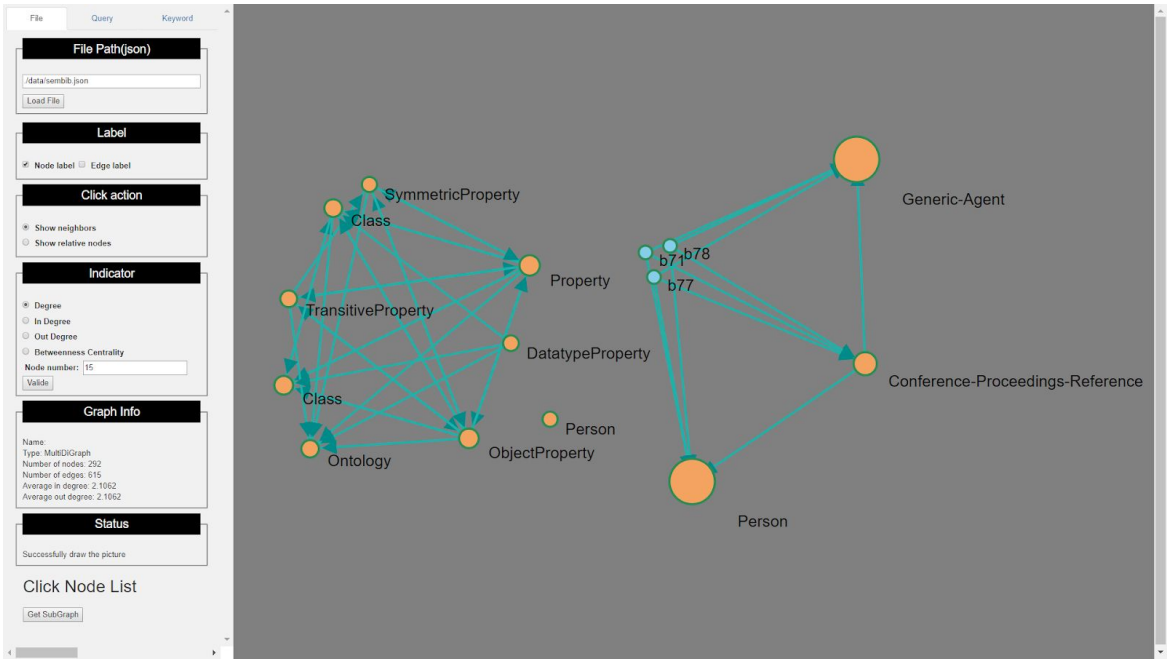
Nous avons réalisé trois modes de visualisation. Dans notre cas d'étude, il n'y a pas beaucoup d'informations contenues dans les données. Nous n'avons que les données comme triplets pour construire un réseau. La clé de la visualisation est donc de savoir

comment gérer un très grand nombre de points, afin que les utilisateurs puissent visualiser et comprendre les données plus efficacement.

Dans le premier mode, l'utilisateur renseigne un lien vers un graphe résumé. Il n'a aucune connaissance de ce graphe et veut explorer ce graphe pour trouver des liens intéressants. Dans le deuxième mode de visualisation, l'utilisateur va entrer des mots clefs et il va pouvoir visualiser un graphe résumé du graphe RDF d'origine qui dépend des mots clefs. Dans le troisième mode de visualisation, la visualisation portera sur les résultats de l'algorithme vu précédemment.



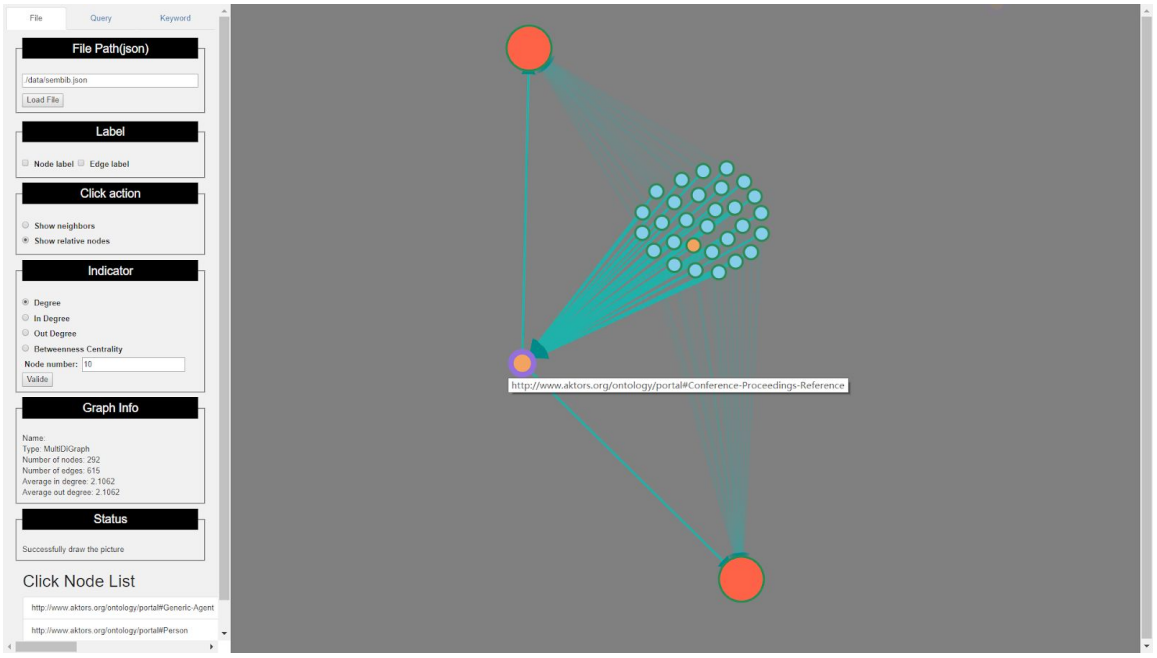
Quelque soit le mode de visualisation, l'utilisateur peut réaliser les fonctions suivantes. l'utilisateur peut entrer le nombre de points à afficher. Les utilisateurs ne peuvent entrer que des chiffres, sinon les lettres seront effacées. Si l'utilisateur entre 0, le graph complet est affiché. L'utilisateur peut alors choisir de générer le graph en fonction de « degree » « indegree » « outdegree » ou « betweenness centrality ». Si l'utilisateur sélectionne « degree » les points avec le plus haut degré en fonction du numéro entré par l'utilisateur seront affichés. Ensuite, nous découvrons les liens directs et indirects entre ces points et générons un nouveau graph. En outre, l'utilisateur peut également choisir d'afficher les étiquettes (label) des sommets. Ces étiquettes sont enregistrées dans les données d'origine sous forme d'URL, elles sont donc très longues. Nous avons effectué le traitement afin que le contenu de l'étiquette ne soit pas affiché assez court. Pour les points affichés, nous les divisons en deux catégories selon la nature des points et les représentons en différentes couleurs.



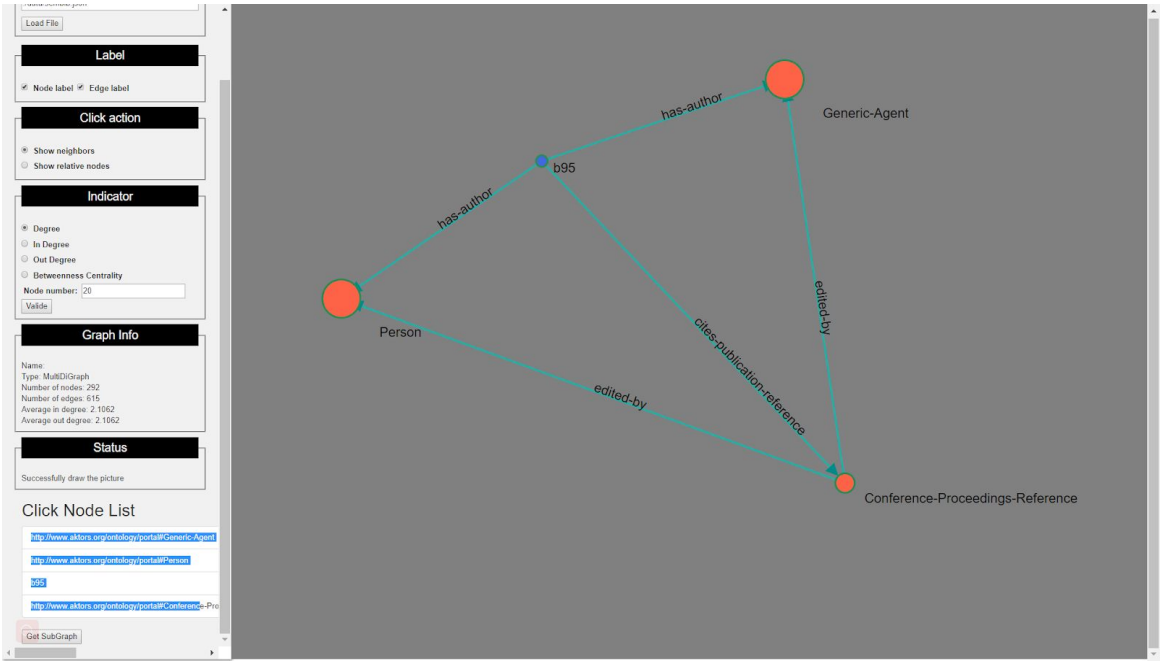
Ici, le graphe de donnée représenté correspond au graphe résumé de sembib en utilisant la méthode présentée dans la partie “Déroulement des recherches”. L’utilisateur a choisi d'afficher les labels des sommets affichés. Afin d’obtenir une meilleure visibilité et une meilleure interprétation du graphe, ce ne sont pas les URI en entier qui sont affichés mais seulement les suffixes. De plus, il a indiqué qu’il souhaite afficher seulement les 15 sommets du graphe avec les degrés les plus importants. Le critère d’importance d’un sommet peut être changé dans l’encadré “Indicator”. Il est possible qu’il y est plus de sommets affichés que le nombre renseigné. En effet, notre algorithme va afficher aussi les sommets qui font partie d’un chemin menant d’un sommet d’importance à un autre. Cela est fait afin de ne pas afficher des sommets seuls alors qu’ils sont en fait connectés au reste du graphe. Les sommets du graphe sont représentés avec des couleurs différentes selon leur type (B-nodes ou bien URI). Dans l'encadré “graph info”, on peut voir des informations concernant le graphe dans sa totalité ce qui permet de se rendre compte de sa taille.

Si l'utilisateur déplace la souris sur un point, les arêtes et les sommets qui ne sont pas voisin de ce sommet deviendront légèrement transparents. Cela permet de mieux visualiser les relations de ce noeud.

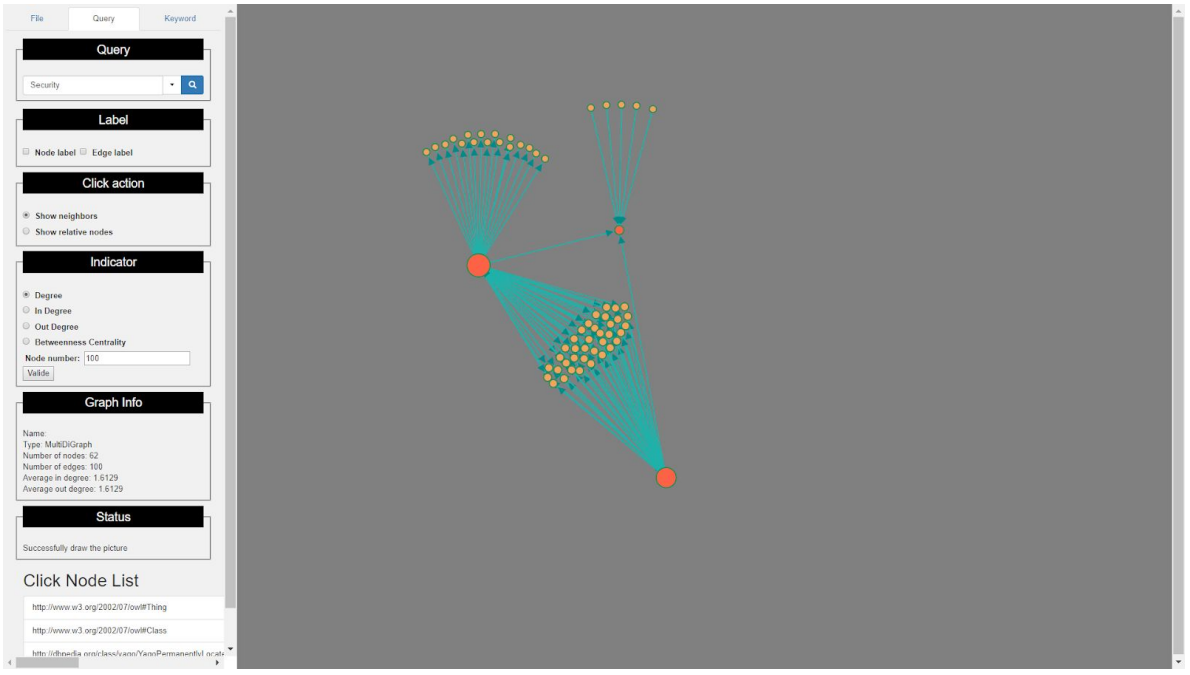
Puisque nous avons utilisé Force Layout en D3 pour visualiser, le graphe est dynamique. L'utilisateur peut fixer la position d’un point en cliquant sur ce point et en le faisant glisser. Si il souhaite libérer le point, il peut cliquer dessus. La couleur du point fixe devient considérablement plus sombre. De plus, les utilisateurs peuvent explorer de nouveaux points en plus des points déjà affichés sur le graph. En faisant ctrl + clique sur un point, tous les voisins (fils) de ce point seront affichés ou bien tous les noeuds (fils + parents) reliés à ce point selon l’option choisi dans l'encadré “click action”. Les points cliqués (points explorés). Le « stroke » du point cliqué devient épaisse et décolorée. Cliquez à nouveau pour restaurer. En outre, nous avons également ajouté la fonction de zoomer.



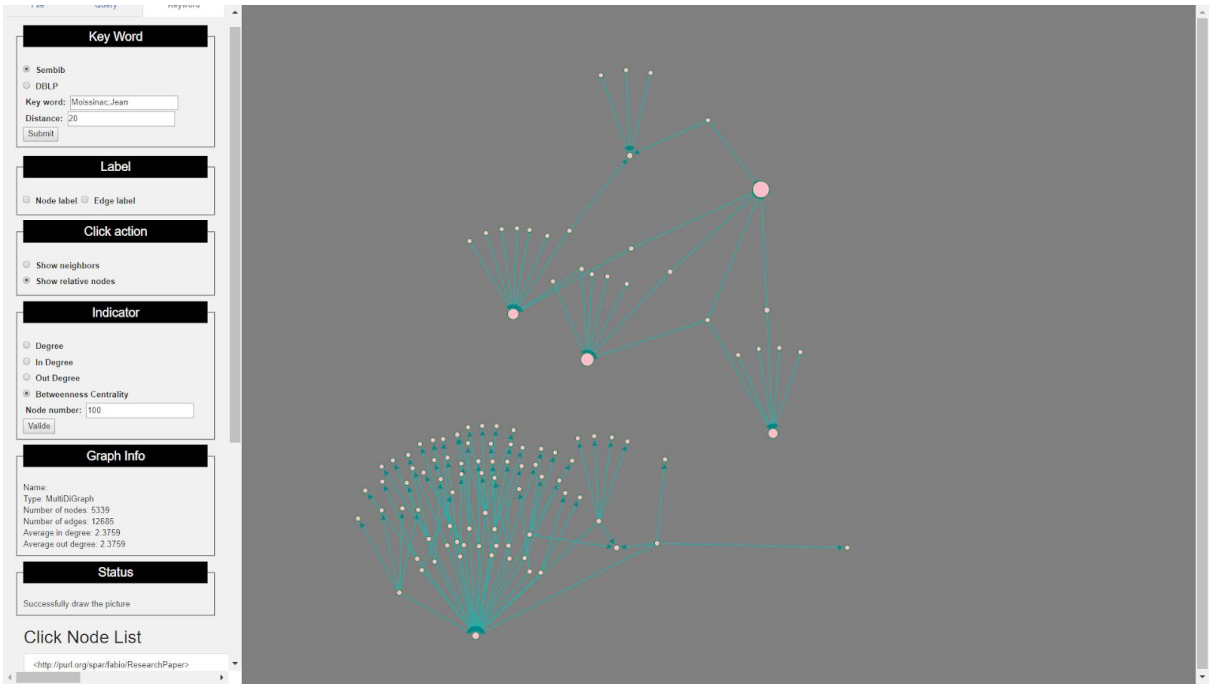
Ici, l'utilisateur a choisi de visualiser les 10 sommets les plus importants, il clique et glisse les deux sommets les plus grands afin de les fixer. On voit bien que les sommets sont fixés car leur couleur devient plus sombre. Ensuite, on voit qu'un des sommets est entouré de violet. Cela signifie que l'utilisateur a ctrl + clique ce sommet. Dans l'encadré "click action", il a choisi d'afficher les "relatives nodes". Cela signifie que les sommets parents et enfants du sommet sélectionné avec ctrl + clique vont être affichés. De plus comme l'utilisateur a placé sa souris sur le sommet cliqué, on voit que les sommets non connectés à ce sommet deviennent légèrement transparents. Cela permet de mieux visualiser les relations qu'entretient ce sommet avec les autres sommets du graphe. Par ailleur, un étiquette est affiché et montre l'URI associé au sommet cliqué.



Ici, comme on a présenté, les sommets oranges sont fixes et sélectionnés, les identifiants de ces sommets sont enregistrés et affichés à gauche en bas dans “click node list”. si l’utilisateur veut voir plus clair, il peut cliquer sur le bouton “Get SubGraph” en bas, seulement les sommets sélectionnés et les sommets nécessaires pour relier ces sommets restent, les autres vont disparaître.



Ici l’utilisateur à choisi le mode Query. Dans ce mode, les requêtes sont effectuées par la construction d’un URL qui va interroger directement la base de donnée choisie. L’utilisateur va entrer des mots clefs. Une requête va être construite afin de créer un graphe résumé du graphe interrogé. Cependant, comme les graphes résumé peuvent être potentiellement très gros, la requête effectuée ici va essayer de créer un graphe résumé en fonction des mots clés donnés afin de restreindre le résultat. Le processus sera similaire à celui expliqué dans la partie “Déroulement des recherches” à la différences que les classes qui vont être sélectionnées devront être des classes reliées à des entités qui contiennent les mots clefs. Malgré ces mesures, les requêtes restent souvent longues car il n’y a aucune phase de preprocessing mais c’est justement cette piste que nous souhaitions explorer ici afin d’essayer d’avoir du temps réel notamment pour pouvoir interroger les graphes gigantesques et en constante évolution tels que dBpedia. L’algorithme présent dans l’onglet keyword est beaucoup plus efficace car il y une phase de pré processing mais elle n’est pas réalisable sur des graphes géants comme dBpedia.



Ici, l'utilisateur a entré deux mots clefs "Jean" et "Moissinac" qui sont souvent une partie de noms des professeurs. Peut-être que l'utilisateur veut trouver des informations et des liens entre des professeurs qui s'appellent "Moissinac" ou "Jean". Le point au centre au-dessous avec le plus de liens est "Jean-Claude-Moissinac" qui match le mieux avec les deux mots clefs. Quelques centres au-dessus sont les professeurs dont leurs noms contiennent un des deux mots clefs. L'algorithme comme déjà présente dans section précédente.