Sorbonne Université
Faculté de Science et d'ingénierie
Département Informatique

Rapport du PSTL

Informatique

Spécialité :
Science et Technologie Logiciel

Thème

Génération et réparation d'instances pour JSON Schema

Encadré par

- Mohammed-Amine Baazizi
- Lyes Attouche

Réalisé par

- Tabellout Salim
- Tabellout Yanis
- Bouzourine Hichem

Soutenu le : **DD/MM/2024**

TABLE DES MATIÈRES

TABLE DES FIGURES

CHAPITRE 1

ETAT DE L'ART

1. JSON Schema

Le JSON Schema [1] est une norme permettant de décrire la structure et les contraintes des données au format JSON (JavaScript Object Notation). Il spécifie la manière dont les données JSON doivent être organisées, les types de données autorisés, les valeurs par défaut, etc.

1.1. Utilité du JSON Schema

1. **Validation des données :** Il permet de valider si une instance JSON est conforme à un schéma prédéfini, assurant ainsi la qualité et la cohérence des données.
2. **Documentation :** En décrivant la structure des données attendues, le JSON Schema sert également de documentation explicite pour les utilisateurs et les développeurs.
3. **Communication :** En partageant un schéma, différentes parties prenantes peuvent avoir une compréhension commune de la structure des données, facilitant ainsi l'échange d'informations.
4. **Génération de données de test :** Il peut être utilisé pour générer des jeux de données de test conformes au schéma, ce qui est utile lors de la phase de développement et de tests.

1.2. Intégration avec les Objectifs du Projet

Dans le cadre du projet, les objectifs visent la génération et la correction d'instances JSON conformes à un schéma initial, tout en minimisant les modifications nécessaires.

1. **Validation initiale :** Les générateurs d'instances identifiés dans l'objectif 1 produisent des données JSON à partir des schémas. La première étape consiste à valider

ces instances par rapport au JSON Schema, identifiant ainsi les non-conformités.

2. **Réparation des instances :** L'objectif global du projet est de développer des approches de réparation permettant de minimiser les modifications nécessaires pour rendre une instance non conforme conforme au schéma initial.
3. **Analyse des erreurs de validation :** L'objectif 4 consiste à étudier le lien entre les erreurs de validation, détectées à l'étape 1, et la distance d'édition entre les instances non conformes et l'instance valide. Cette analyse contribue à une compréhension approfondie des types d'erreurs et guide le processus de réparation.

2. Génération d'instances

La génération d'instance pour un schéma json est une tâche complexe du à la forme non algébrique du JSON schema. Un langage est dit "algébrique" quand l'application et la séméntique de ses opérateurs dépéndent de la sémantique de leurs opérandes, pour json nous avons l'interaction syntaxique ainsi que sémantique.[2]

3. La similarité des documents JSON

La similarité des documents JSON est une mesure de la similarité entre deux documents JSON. Elle est généralement utilisée pour comparer des documents JSON qui représentent des objets ou des données similaires.

3.1. Approches existantes

Une des approches existantes pour calculer la similarité des documents JSON est :

- **Approche top-down [3] :** Cet approche top-down pour un comparateur de similarité dans le contexte JSON consiste à examiner la similarité entre deux structures JSON en commençant par les éléments les plus généraux et en descendant progressivement vers les détails spécifiques. Cela implique une comparaison basée sur la hiérarchie des éléments plutôt que sur les valeurs individuelles. ensuite les valeurs des propriétés et des éléments des deux documents.

3.2. Limitations :

1. **La structure du document est ignorée :** les approches top-down ignorent la structure du document, ce qui peut conduire à des résultats inexacts car dans un document JSON, on retrouve des éléments ordonnées (*array*), et non ordonnées (*objects*).
2. **Aucune garantie de qualité n'est donnée :** les approches existantes ne fournissent généralement aucune garantie de qualité pour leurs résultats.

4. JEDI

JEDI [4] est un algorithme de calcul de la similarité entre deux documents JSON. Il fonctionne en comparant les deux documents en tant qu'arbres en prenant en compte la structure du document (*ordonnée vs non ordonnée*). La similarité entre les deux documents est définie comme le nombre minimum d'opérations d'édition (Ajout, Suppression, Modification) nécessaires pour transformer un arbre en l'autre.

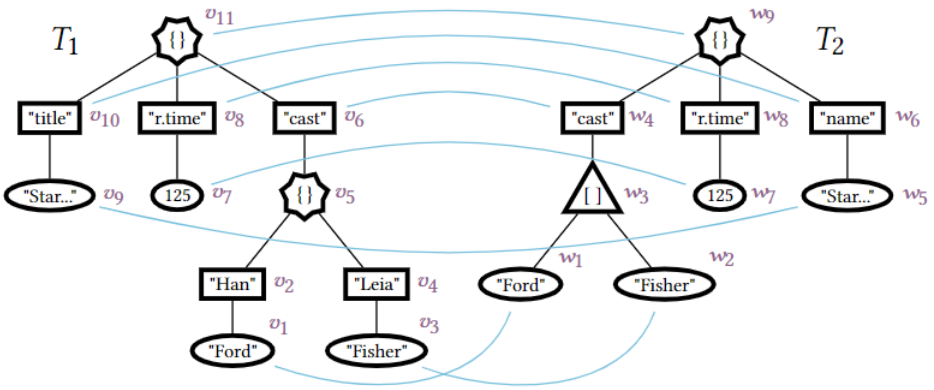


FIGURE 1.1 – Exemple de nombre d'opération de transformation [4]

4.1. Représentation d'Arbres JSON

JEDI introduit la notion d'arbre JSON qui est une représentation arborescente d'un document où chaque valeur de l'arbre est représenté par un noeud, nous permettant ainsi d'exploiter certaines propriétés des arbres qui nous sera utile par la suite. La figure 1.2 est un exemple sur cette notion.

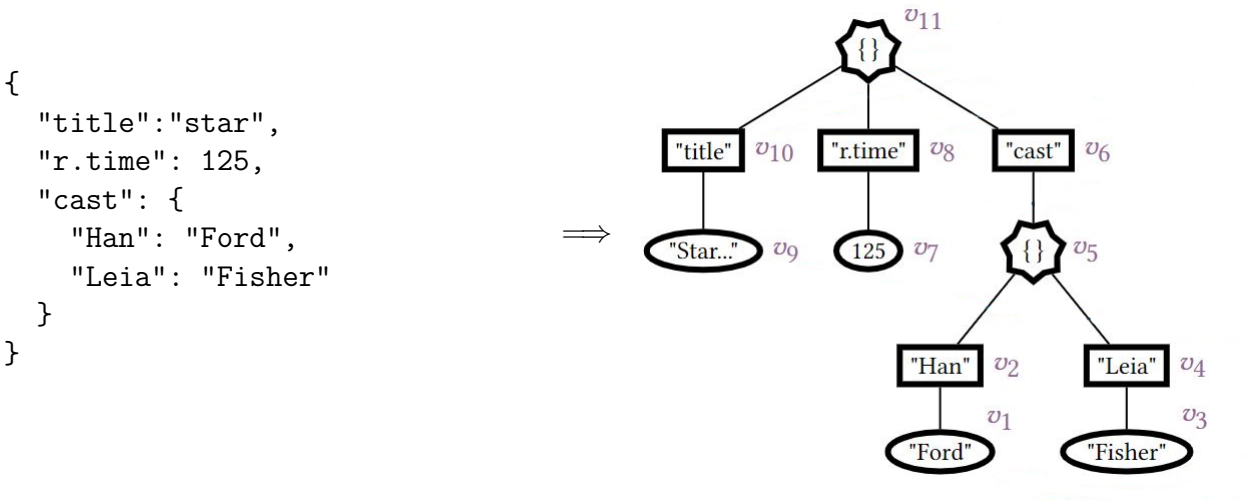


FIGURE 1.2 – Transformation d'un document json en arbre Json

4.2. Distance d'édition (Tree Edit Distance : TED)

JEDI permet de calculer la **distance d'édition** entre **deux arbres JSON**. On définit **TED** comme étant le nombre d'opération minimale pour transformer un arbre T_1 à l'arbre T_2. Ces opérations sont : *ajout, suppression, renomer un noeud* et qui représentent les opérations permettant le **Json Edit Mapping**, c'est à dire faire correspondre chaque noeud du premier arbre vers le deuxième soit :

- Suppression : Les noeuds de T_1 qui ne figurent pas dans T_2 sont supprimés
- Ajout : Les noeuds de T_2 qui ne figurent pas dans T_1 sont ajoutés
- Renommer : Les noeuds de T_1 qui figurent dans T_2 sont renommés

Chaque opération est associé à un coût, sauf le cas de renommer le même noeud [1]

4.3. Distance et Matrice d'édition

Jedi permet de construire une matrice appelé **Matrice d'édition** qui sauvegarde la distance d'édition entre chaque noeuds de T_1 avec les noeuds de T_2, la figure suivante illustre la matrice de l'exemple précédent :

1. Le coût de renommer $v8$ en $w8$ est nulle 1.1 car il s'agit du même noeud

dt	ϵ	w_1	w_2	...	w_6	w_9
ϵ	0	1	1	...	2	9
v_9	1	1	1	...	1	8
v_{10}	2	2	2	...	1	8
⋮	⋮	⋮	⋮	⋱	⋮	⋮
v_6	6	5	5	...	6	8
v_{11}	11	10	10	...	10	5

FIGURE 1.3 – Matrice d'édition entre T_1 et T_2

Exemple :

```
{
    "tree":[
        {"height" : 300},
        {"width" : 20}
    ]
}
```

(a) Json T_1

```
{
    "tree":[
        300,
        20
    ]
}
```

(b) Json T_2

dt	{}	tree	[ ]	300	20
{}	4	5	6	8	8
tree	5	4	5	7	7
[ ]	6	5	4	6	6
{}	3	2	2	2	2
height	3	2	1	1	1
300	4	3	2	0	0
{}	3	2	2	2	2
width	3	2	1	1	1
20	4	3	2	0	0

FIGURE 1.4 – Matrice d'édition

Remarque *Pour Jedi on ne sauvegarde pas les opérations qu'on fait mais plutôt le coût de modification.*

5. Validation d'un schema

La **validation** du JSON Schema est un processus par lequel les données JSON sont vérifiées par rapport à un schéma JSON spécifié pour garantir leur **conformité aux règles** et aux contraintes définies dans ce schéma comme les règles de typage[5]. Le JSON Schema fournit une méthode standardisée pour décrire la structure attendue des données

JSON, y compris les types de données, les propriétés requises, les valeurs autorisées, les formats de données et les relations entre les différentes parties des données. [6]

Lorsqu'une validation JSON Schema est effectuée, les données JSON sont *comparées* au schéma correspondant, et toute violation des règles spécifiées dans le schéma est *détectée et signalée*.

La validation d'un document par rapport au schéma est toujours en **PTIME** et peut être résolu en temps linéaire tant que le schéma n'utilise pas de *uniqueItems*.

5.1. validation d'un document en PTIME

Nous traitons le document restriction par restriction, tout en vérifiant la conformité au sous-schéma correspondant dans S. Le temps d'exécution est **linéaire** car la correspondance à chaque mot-clé du schéma JSON peut être vérifiée en temps linéaire (sauf pour les éléments uniques).

5.2. Validation d'un document en PTIME-hard

On fait la même chose mais on vérifie aussi que les éléments d'un tableau J sont uniques, d'abord en *triant* le tableau J.

La preuve est par réduction du problème de la valeur du circuit monotone

Exemples de Grammaire JSON Schema

- **JSDoc :=** (defs,) ? JSch
- **Defs :=** "definitions :string :JSch (,string : JSch)*
- **JSch :=** strSch | numSch |intSch | objSch | arrSch | refSch | not | allOf | anyOf | enum
- **not :=** "not" : JSch

CHAPITRE 2

ETUDE EXPÉRIMENTALE

1. Dataset

Le dataset a déjà été créé et comprend environ 6000 schémas. Cependant, tous les témoins n'ont pas été correctement générés, c'est pourquoi nous ne conservons que la moitié de ces schémas, soit un total de 3000 schémas.

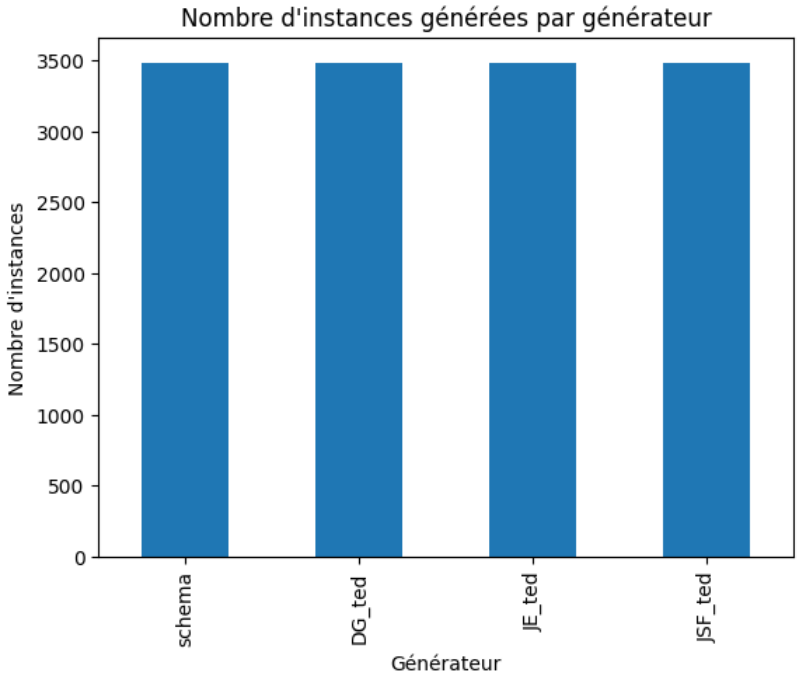


FIGURE 2.1 – Nombre d'instance par générateur

2. Génération d'instance Valide

La génération d'instances JSON Schema vise à créer des ensembles de données valides à partir d'un schéma JSON défini. Ce processus est crucial pour divers cas d'utilisation, tels que la création de jeux de test, le remplissage de bases de données et l'exploration de l'espace de solutions défini par le schéma [7]. Pour celà on dispose des schémas, ainsi que des instances correctes, et nous utilisons trois générateurs d'instance, ces instances seront appelé witness :

- **json-schema-faker (JSF) :** pour une génération rapide et simple de données fictives[8].
- **json-everything (JE) :** écrite en C#, est extension de **System.Text.Json**, limitée en termes d'expressivité sur la partie JSON[9].
- **json-data-generator (DG) :** pour une prise en charge complète de JSON Schema Draft 7 et la génération de données aléatoires[10].

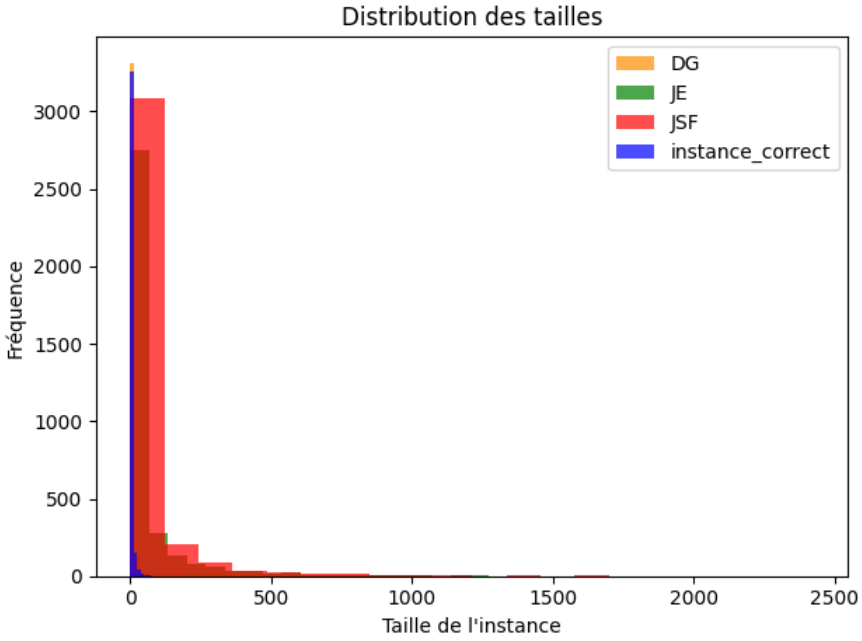
Le dataset contenant les witness a été déjà construit précédemment, donc on mènera nos tests dessus

3. Distance d'édition et erreurs des instances

Une première approche serait de voir si il existe un lien entre la distance d'édition et le nombre d'erreurs pour chaque witness, celà en calculant la distance d'édition entre chaque instance correct et les witness de chaque générateurs

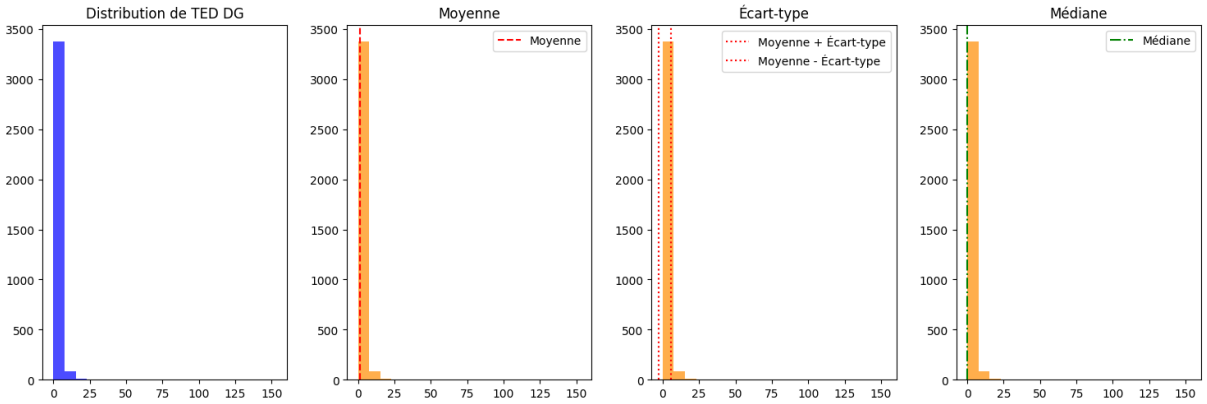
3.1. Distribution des tailles

Pour cette partie on s'intéresse à la distribution des tailles des instances corrects ainsi que les witness.



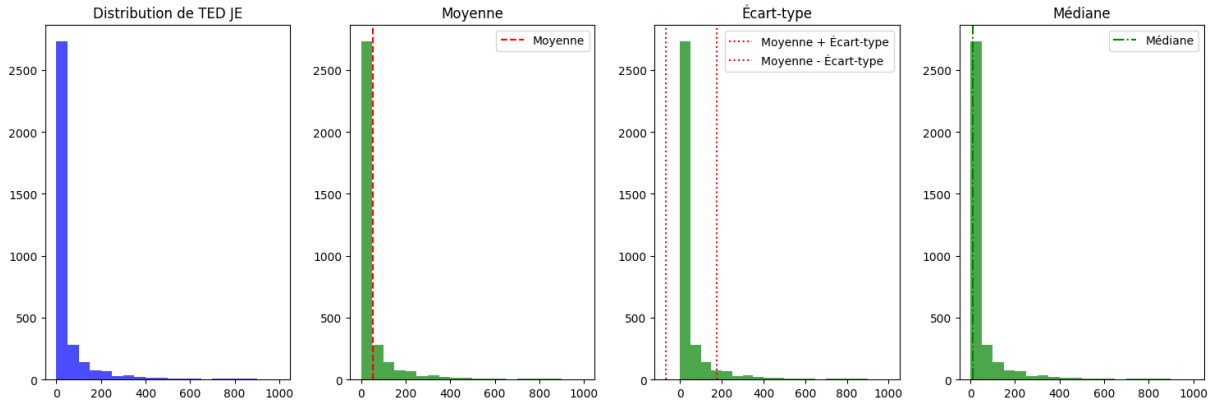
3.2. Distribution des distances d'édition

On s'intéresse dans cette partie à savoir connaitre certaines statistiques concernant les distances d'édition pour chacun des générateurs

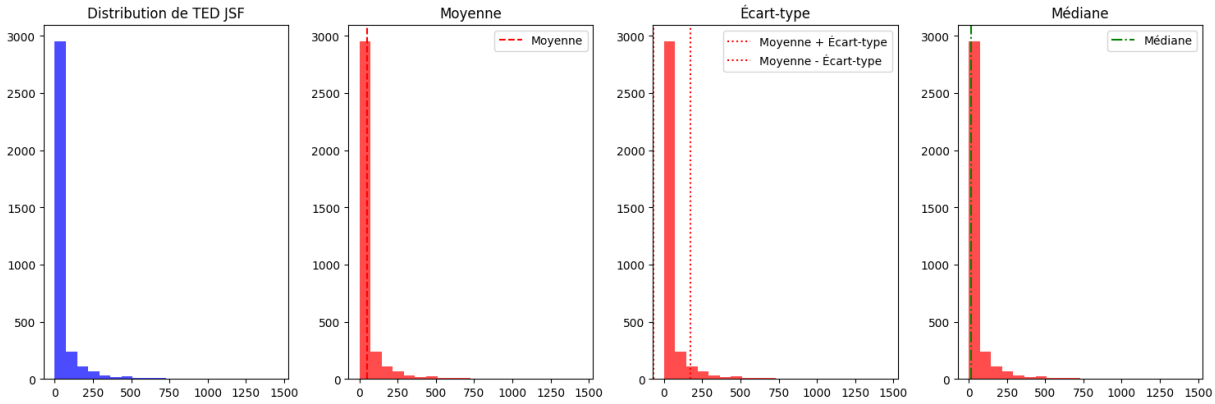


(a) Distrubtion TED DG

FIGURE 2.2 – Distribution des TED



(b) Distrubtion TED JE



(c) Distrubtion TED JSF

FIGURE 2.2 – Distribution des tailles des instances

3.3. Relation entre distance d'édition et nombre d'erreurs

On s'intéresse dans cette partie à savoir connaitre certaines statistiques concernant les distances d'édition pour chacun des générateurs

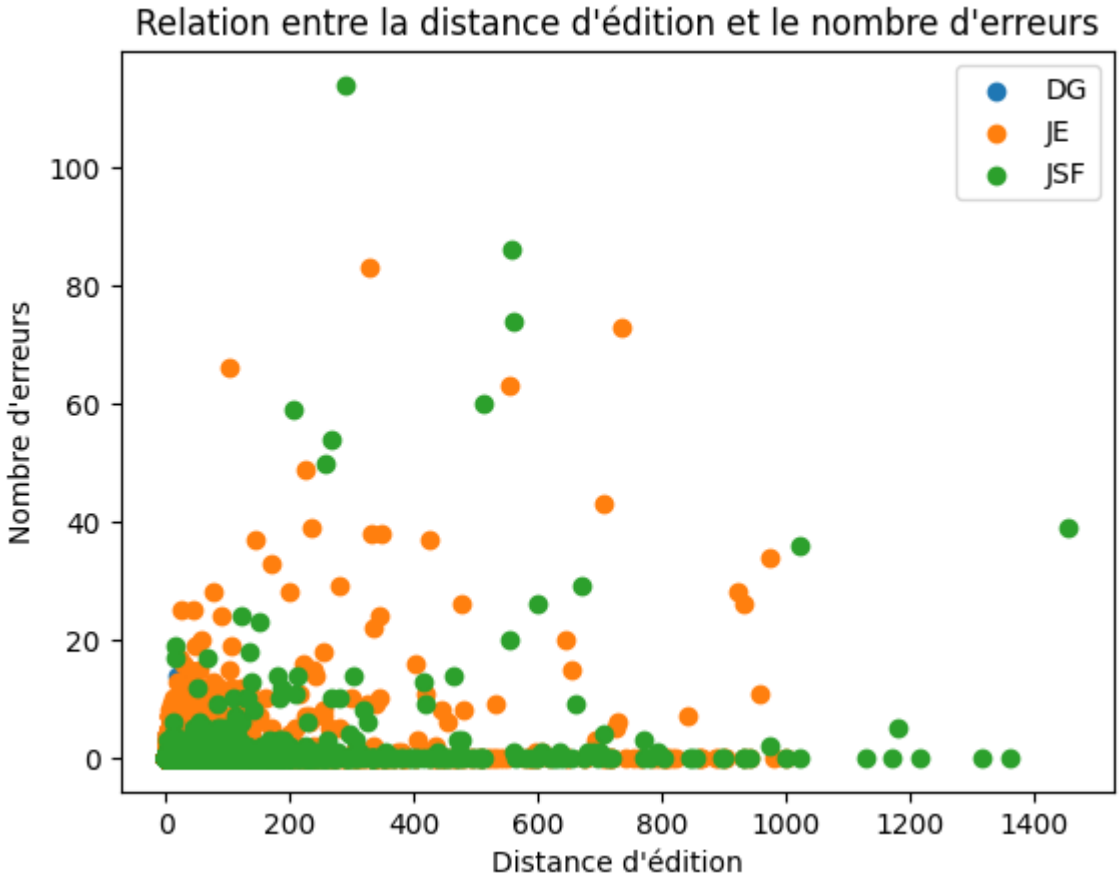
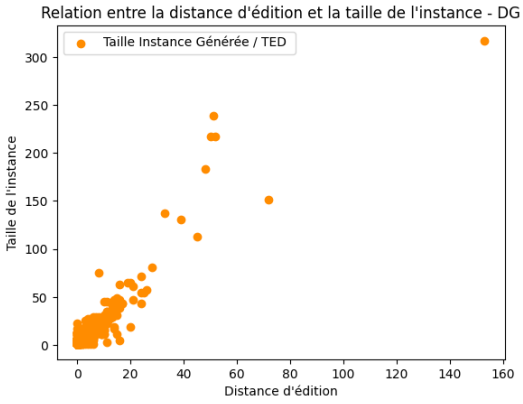
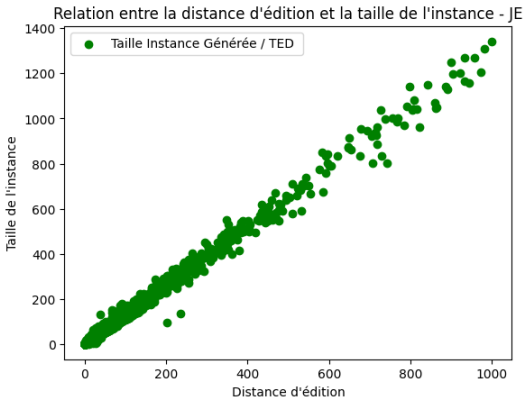


FIGURE 2.3 – Nombre d'erreurs et TED

On s'intéresse aussi à la relation entre la taille des witness par rapport à leurs TED (de l'instance correct)



(a) Taille d'instances et TED (DG)



(b) Taille d'instances et TED (JE)

FIGURE 2.4 – Distribution des TED

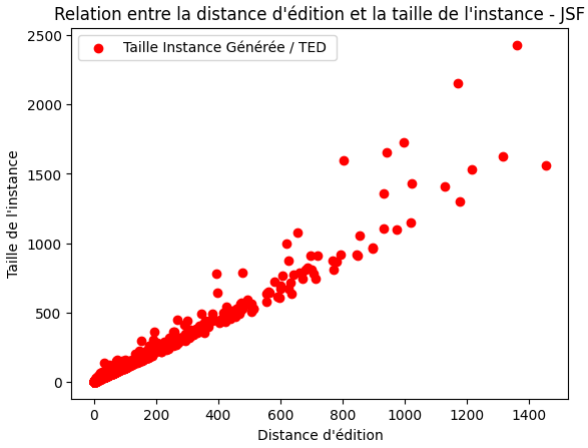


FIGURE 2.4 – Taille d'instances et TED (JSF)

BIBLIOGRAPHIE

[1] Json schema. https://json-schema.org.

[2] Mohamed Amine Baazizi et al. Not elimination and witness generation for json schema. bda, 2020.

[3] Json similarity comparitor. https://github.com/Geo3ngel/JSON-Similarity-comparitor.

[4] Thomas Hütter, Nikolaus Augsten, Christoph Kirsch, Michael Carey, and Chen Li. Jedi : These aren't the json documents you're looking for ? pages 1584–1597, June 2022.

[5] Lyes Attouche, Mohamed-Amine Baazizi, Dario Colazzo, Giorgio Ghelli, Carlo Sartiani, and Stefanie Scherzinger. Validation of modern json schema : Formalization and complexity. *Proceedings of the ACM on Programming Languages*, 8 :1451–1481, January 2024.

[6] Felipe Pezoa, Juan Reutter, Fernando Suarez, Martin Ugarte, and Domagoj Vrgoč. Foundations of json schema. pages 263–273, April 2016.

[7] Benchmarking de solutions optimistes pour génération de données test à partir de json schema. Sorbonne Universite - Faculté de Science et ingénierie Master Informatique parcours STL, 2023.

[8] json-schema-faker. https://github.com/json-schema-faker/jsonschema-faker, 2023.

[9] json-everything. https://github.com/gregsdennis/json-everything, 2023.

[10] json-data-generator. https://github.com/jimblackler/jsongenerator.