



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

1	Etat de l'art	1
1.	JSON Schema	1
1.1.	Utilité du JSON Schema	1
1.2.	Intégration avec les Objectifs du Projet	1
2.	La similarité des documents JSON	2
2.1.	Approches existantes	2
2.2.	Limitations :	2
3.	JEDI	2
3.1.	Représentation d'arbres Json	3
3.2.	Principe de calcul de similarités de JEDI	3
3.2.1.	QuickJedi	4
3.3.	Distance et Matrice d'édition	4
3.3.1.	Distance d'édition	4
4.	Validation d'un schema	4
4.1.	validation d'un document en PTIME	5
4.2.	Validation d'un document en PTIME-hard	5
2	Etude expérimentale	6
1.	Génération d'instance Valide	6
1.1.	Bibliothèques existantes	6
2.	Approche sur les générateurs	6
2.1.	Résultats	7

TABLE DES FIGURES

1.1	Exemple de nombre d'opération de transformation[1]	3
1.2	Transformation d'un document json en arbre Json	3
1.3	Explication de l'algorithme JEDI[1]	4

CHAPITRE 1

ETAT DE L'ART

1. JSON Schema

Le JSON Schema [2] 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. 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.

2.1. 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.

2.2. 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.
2. **Aucune garantie de qualité n'est donnée** : les approches existantes ne fournissent généralement aucune garantie de qualité pour leurs résultats.

3. JEDI

JEDI [1] est un algorithme de calcul de la similarité entre deux documents JSON. Il fonctionne en comparant les deux documents en tant qu'arbres. 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.

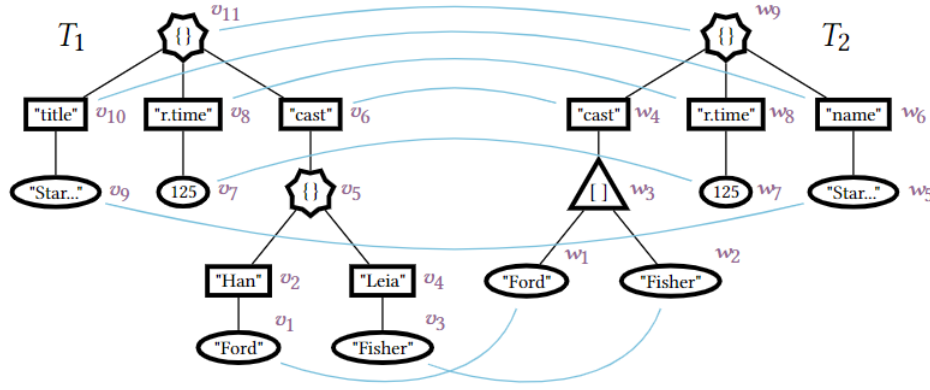


FIGURE 1.1 – Exemple de nombre d'opération de transformation[1]

3.1. Représentation d'arbres Json

JEDI [1] 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.

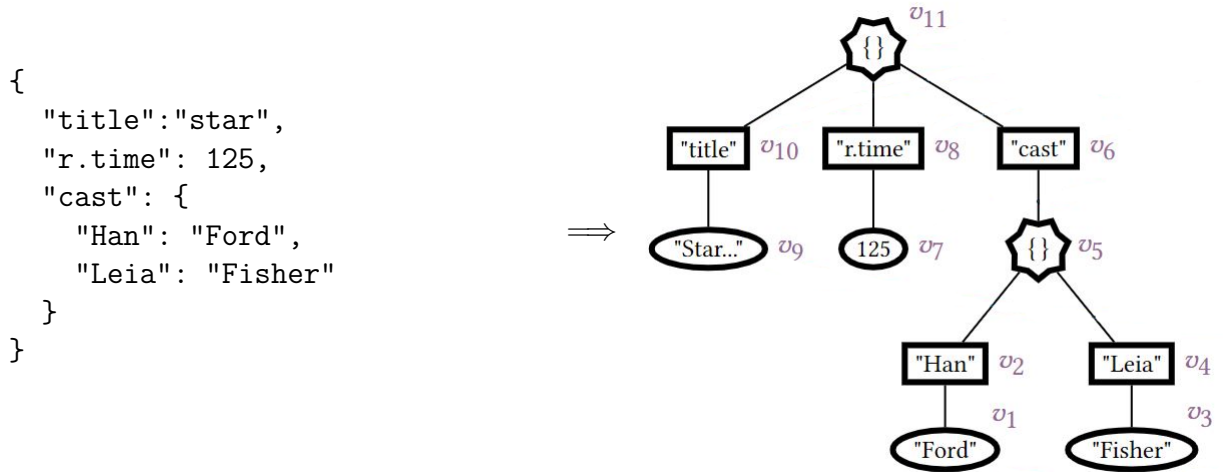


FIGURE 1.2 – Transformation d'un document json en arbre Json

3.2. Principe de calcul de similarités de JEDI

A partir d'un **Threshold T** et des documents d_q , on veut extraire à partir d'une base de données **D** les documents d_i qui sont similaires ($\text{JEDI}(d_q, d_i) < T$). La solution proposée est donc de complexité $O(n^2 \times d \times \log(d))$ avec **n** étant la taille de l'arbre et **d** le degré maximum de l'arbre.

3.2.1. QuickJedi

L'algorithme JEDI est complexe et peut être lent pour traiter de grandes quantités de données. Pour réduire le temps de calcul, il est donc indispensable de filtrer les documents avant de les comparer à l'aide de JEDI. Le filtrage consiste à sélectionner les documents les plus susceptibles d'être similaires à la requête. Une fois les documents filtrés, l'algorithme JEDI peut être utilisé pour calculer la similarité entre les documents candidats.

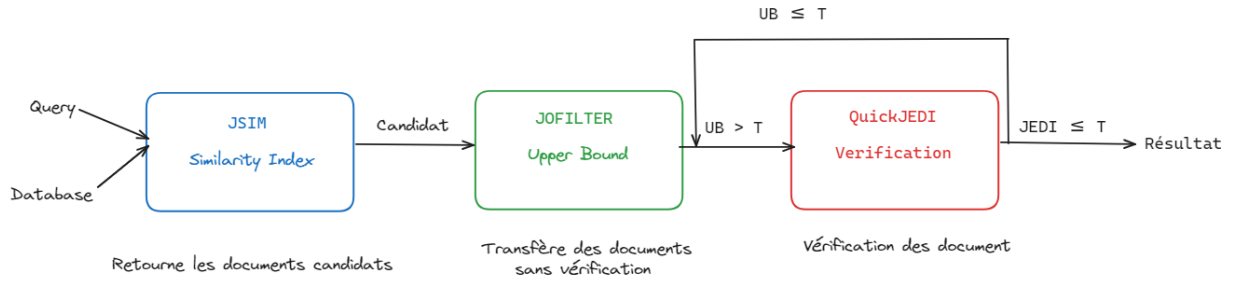


FIGURE 1.3 – Explication de l'algorithme JEDI[1]

3.3. Distance et Matrice d'édition

3.3.1. Distance d'édition

Soit deux documents JSON D_A et D_B , La distance d'édition **TED**¹ est représenté par la formule suivante :

$$\mathbf{TED}(D_A, D_B) = \mathbf{Min}(\{NOP\}) \quad (1.1)$$

- **NOP** : Etant le nombre d'opérations

Les opérations que JEDI utilise sont les suivantes :

1. **Insertion**
2. **Suppression**
3. **Edition**

- **Remarques :**

1. Les operations se font sur les noeuds de l'arbre du document D_A pour le comparer avec l'arbre du document D_B .
2. **JEDI ne permet pas de sauvegarder les opérations**, mais plutôt calculer le nombre d'opération qu'il faut pour transformer un arbre en un autre.

4. 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**

1. Tree Edition Distance[1]

règles et aux contraintes définies dans ce schéma comme les règles de typage[4]. 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. [5]

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*.

4.1. 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).

4.2. 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. 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 [6].

1.1. Bibliothèques existantes

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

2. Approche sur les générateurs

Notre approche consiste à faire une étude expérimentale et trouver si possible, une relation entre le nombre d'erreur et la distance d'édition **TED**. La démarche est donc de prendre le **TED** de **JEDI** et de comparer la distance d'édition d'instance généré par rapport à des instances correctes déjà fournies dans un **dataset**, on calcule pour chaque générateur le nombre d'erreur de chaque instance ainsi que la taille de l'arbre, le résultat nous permettra donc de mieux comprendre comment procéder à notre réparation d'instance.

2.1. Résultats

BIBLIOGRAPHIE

- [1] 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.
- [2] Json schema. <https://json-schema.org>.
- [3] Json similarity comparitor. <https://github.com/Geo3ngel/JSON-Similarity-comparitor>.
- [4] 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.
- [5] Felipe Pezoa, Juan Reutter, Fernando Suarez, Martin Ugarte, and Domagoj Vrgoč. Foundations of json schema. pages 263–273, April 2016.
- [6] 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.
- [7] json-schema-faker. <https://github.com/json-schema-faker/jsonschema-faker>, 2023.
- [8] json-everything. <https://github.com/gregsdennis/json-everything>, 2023.
- [9] json-data-generator. <https://github.com/jimblackler/jsongenerator>.