Projet AI2D

Comparaisons de méthodes de résolutions de Processus de Décision Markoviens sur des instances issues de langage de modélisation

Réalisé par Jiahua LI et Zeyu TAO
Encadré par Emmanuel Hyon et Pierre-Henri Wuillemin

Context - Marmote

- Une bibliothèque pour construire et résoudre des modèles de Markov

- Fonctionnalités fournit:
  - chaînes de Markov
  - processus de décision markoviens
  - ...

- Notre objectif:
  - Comparer l'efficacité des algorithmes de résolution
  - Chercher les limite de la bibliothèque Marmote

Rappelle

- Propriété markovienne
- Chaîne de Markov (MC)
  - Distribution initiale
  - Matrice de transition
- Processus de décision markovien (MDP)
  - Espace d’états
  - Espace d’actions
  - Matrices de transition
  - Matrices de récompense

Modèles Markoviens disponibles dans Marmote

- MC
  - **Objectif:** calculer la distribution stationnaire
  - **Modèle de résolution:**
    - MarkovChain

- MDP
  - **Objectif:** calculer le politique optimal et optimiser les récompenses obtenues
  - **Modèles de résolution:**
    - DiscountedMDP
    - TotalRewardMDP
    - FiniteHorizonMDP
    - AverageMDP

Problématique : Il n’existe aucun benchmark pour tester des différents algorithmes.

- La bibliothèque Marmote ne contient que quelques petites instances créées manuellement.

Cahier des charges

- **Chercher des langages de description et des benchmarks**
- **Choix du langage**
- **Implémentation**
- **Utilisation de cette implémentation: benchmarks**

Recherche des langages de description et benchmark

Recherche des langages de description et benchmark

- **Méthodologie**
    - Recherches avec Google, Google Scholar, ChatGPT
    - Mots-clés: solver MDP, benchmark, description language...
- **Langages étudiés**

  PPDDL, RDDL, JANI, PRISM, MODEST, PGCL, GSPN...
- **première selection**
    - JANI
    - RDDL

Choix du langage

RDDL (Relational Dynamic Influence Diagram Language)

- Planification
- Toutes les variables, telles que l'état et l'action sont paramétrables
- Les probabilités de transition conditionnelles entre les états sont données par un réseau bayésien dynamique (DBN)
- Écriture très compacte, mais parfois difficile à lire et à analyser

JANI (JSON Automata Network Interface)

- Model checking
- Structure de fichier basée sur JSON
- Supporter une variété de classes de modèles stochastiques, notamment:
  - DTMC (Discrete-Time Markov Chains)
  - MDP (discrete-time Markov Decision Processes)
- Les transitions sont données de manière explicite à travers: l'état source, l'état cible, l'action correspondante et la probabilité de transition
- De nombreux outils permettent de traduire des instances issues de différents langages de description (PPDDL, PRISM, Modest ...etc) vers le format JANI

Choix entre les deux langages

- JANI
  - Structure de fichier basée sur JSON: facile à lire, avec des bibliothèques JSON déjà disponibles en Python
  - Benchmarks très riches: supportent les modèles DTMC, MDP ...etc
  - Parser existant (principalement étudié):
    - Storm
  - Il existe un outil permettant de convertir PPDDL en JANI, ainsi que RDDLSim, qui permet de convertir RDDL en PPDDL. Mais ...

- RDDL
  - Parser difficile à implémenter: nécessite une construction complète de l'arbre syntaxique
  - Benchmarks limités: ne supportent que les modèles MDP et POMDP
  - Parsers existants (principalement étudiés):
    - RDDLSim
    - PyRDDLGym

Choix final: JANI

Mais...

il n’y a pas de récompense explicite définie dans la structure de JANI

Une extension de JANI: JANI-R

Nouvelles fonctionnalités:

- type: modèle de résolution
- criterion: critère d'optimisation
- horizon (optionnel): horizon temporel
- gamma (optionnel): facteur d'actualisation
- reward: $r(s_t, a_t, s_{t+1})$, une écriture plus générale et flexible de la fonction de récompense

L'extension JANI-R est conçue pour modéliser uniquement les modèles:

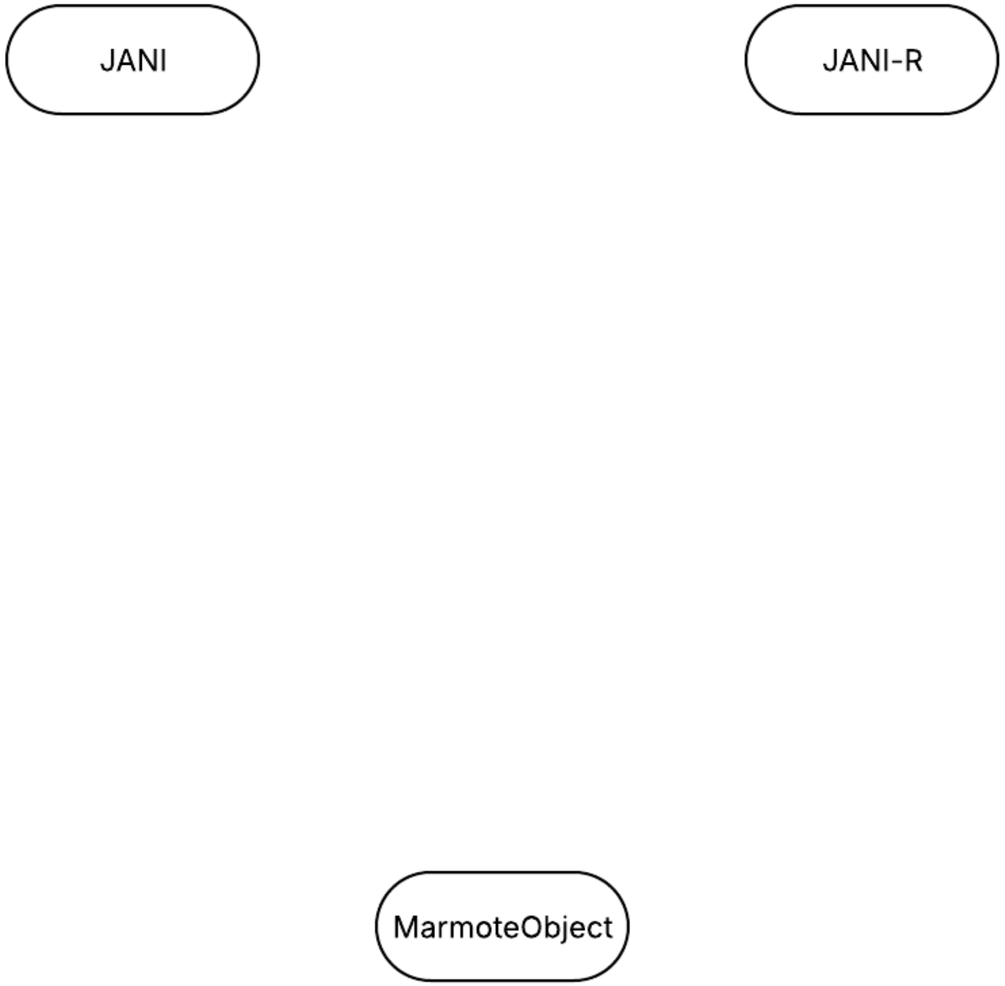
- DTMC
- MDP

```
{
    ...,
    "type": "DiscountedMDP" | "AverageMDP" |
            "TotalRewardMDP" | "FiniteHorizonMDP" |
            "MarkovChain",
    "criterion": "min" | "max",
    "?horizon": <int>,
    "?gamma": <real>,
    "automate": [
    {
        ...,
        "edges: [
        {
            "location": <autamata location>,
            "action": <action>,
            "?guard": { "exp": <expr> },
            "destinations": [
            {
                "location": <autamata location>,
                "?probability": { "exp": <expr> },
                "?reward": { "exp": <expr> },
                "?assignments": [ <assignment>, ... ]
            },
            ...
            ]
        },
        ...
        ]
    }
    ]
}
```

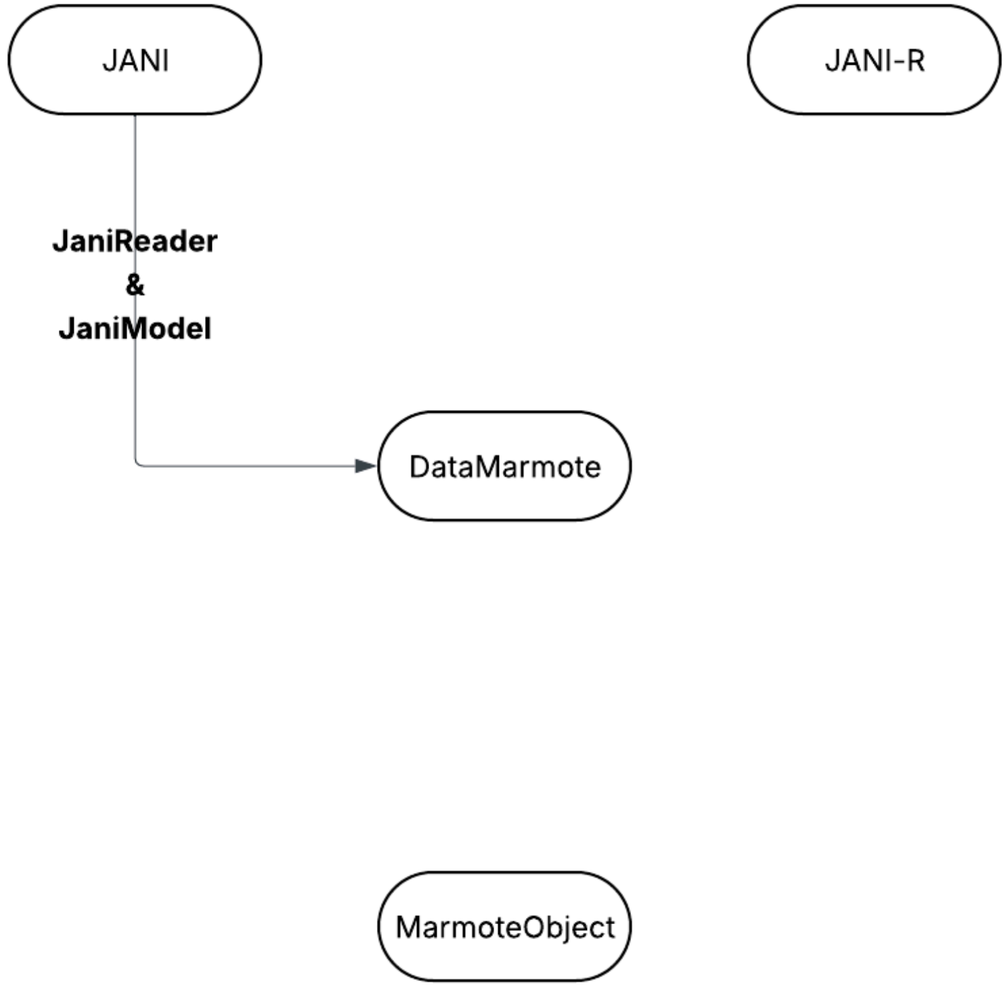
Implémentation

(environ 4k lignes de code)

Composantes de base de l’architecture

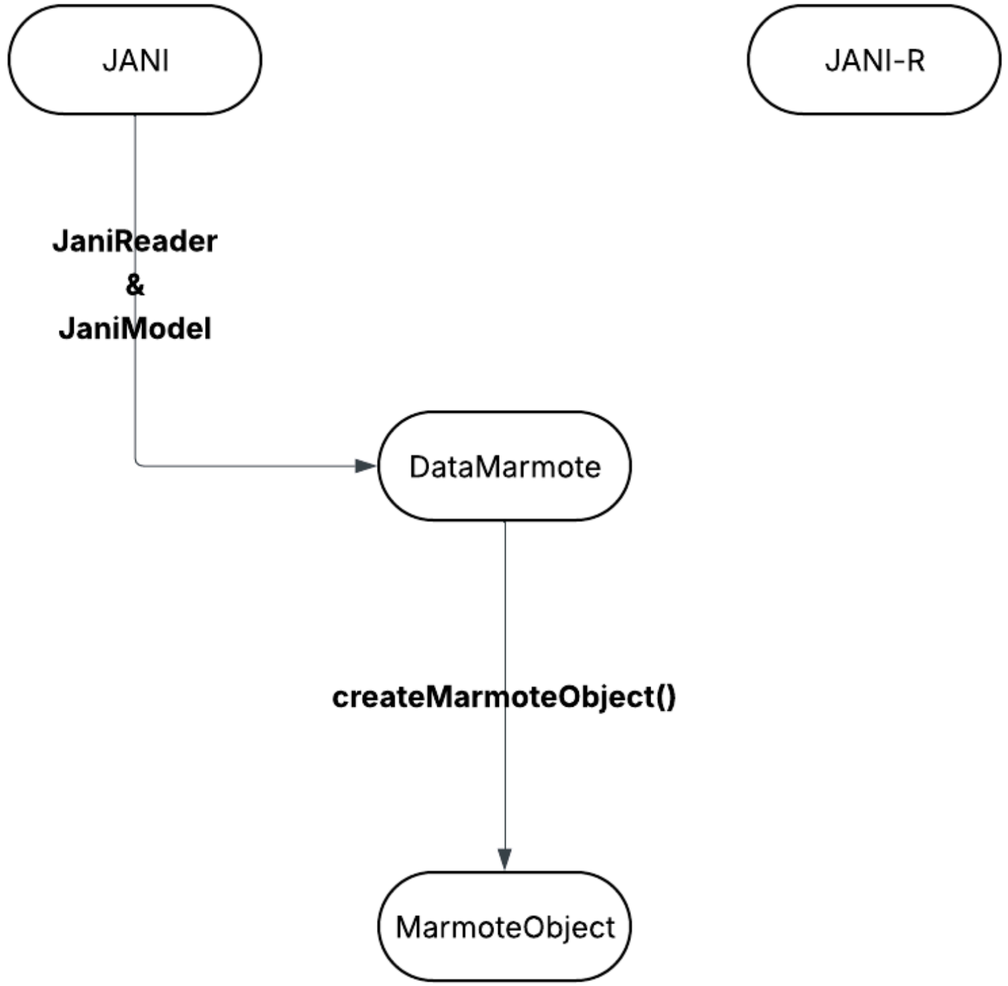


Introduction de l’instance DataMarmote et conversion de JANI vers DataMarmote



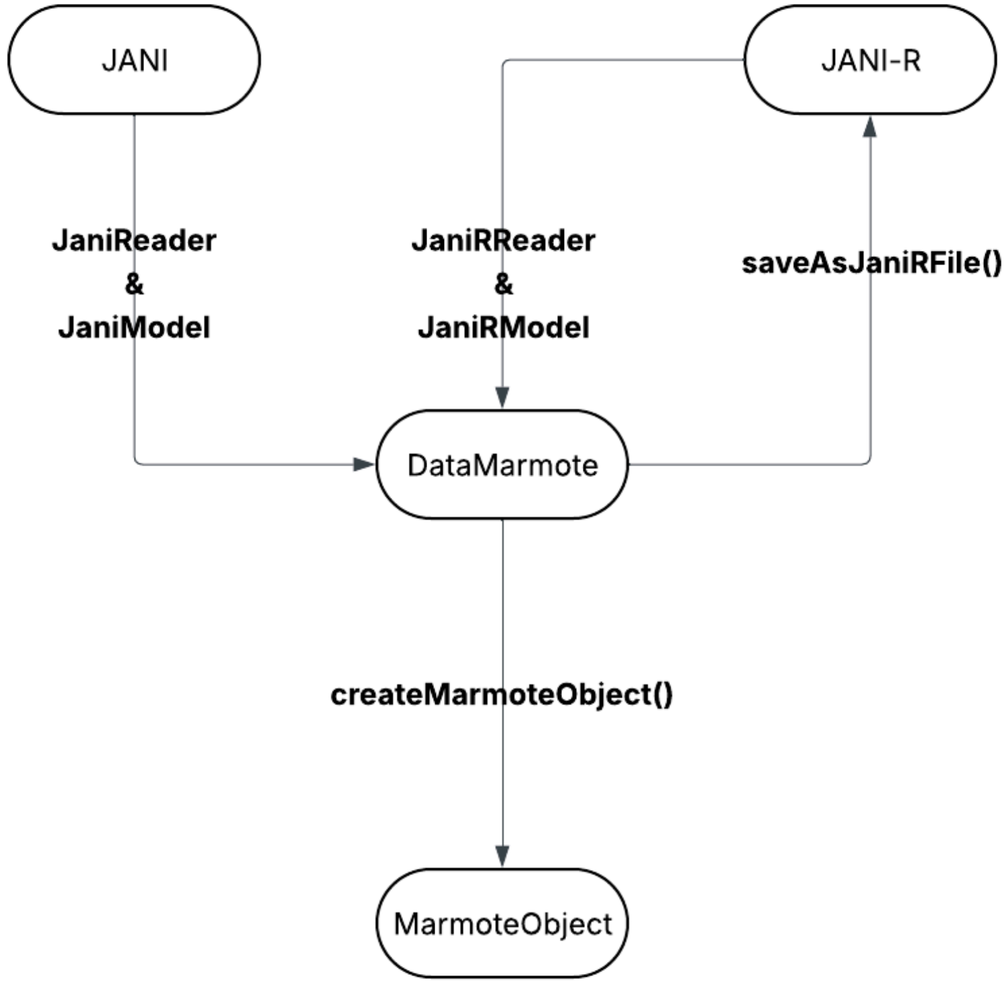
- **DataMarmote:** stocker les données nécessaires à la reconstruction du modèle
- **JaniReader:** lire et parser les fichiers JANI et créer une instance de **JaniModel**
- **JaniModel:**
  - **getMDPData(property name):** retourne les données pour un modèle MDP selon la propriété choisie
  - **getMCData():** retourne les données pour un modèle MC

Création de l’object Marmote



- **DataMarmote:**
  - **createMarmoteObject():** créer une instance de Marmote

Conversion entre JANI-R et DataMarmote



- **DataMarmote:**
  - **saveAsJaniRFile():** exporter les données du modèle dans un fichier au format JANI-R
- **JaniRReader:** lire et parser les fichiers JANI-R et créer une instance de **JaniRModel**
- **JaniRModel:**
  - **getMDPData():** retourne les données pour un modèle MDP
  - **getMCData():** retourne les données pour un modèle MC

Benchmark

Description du benchmark

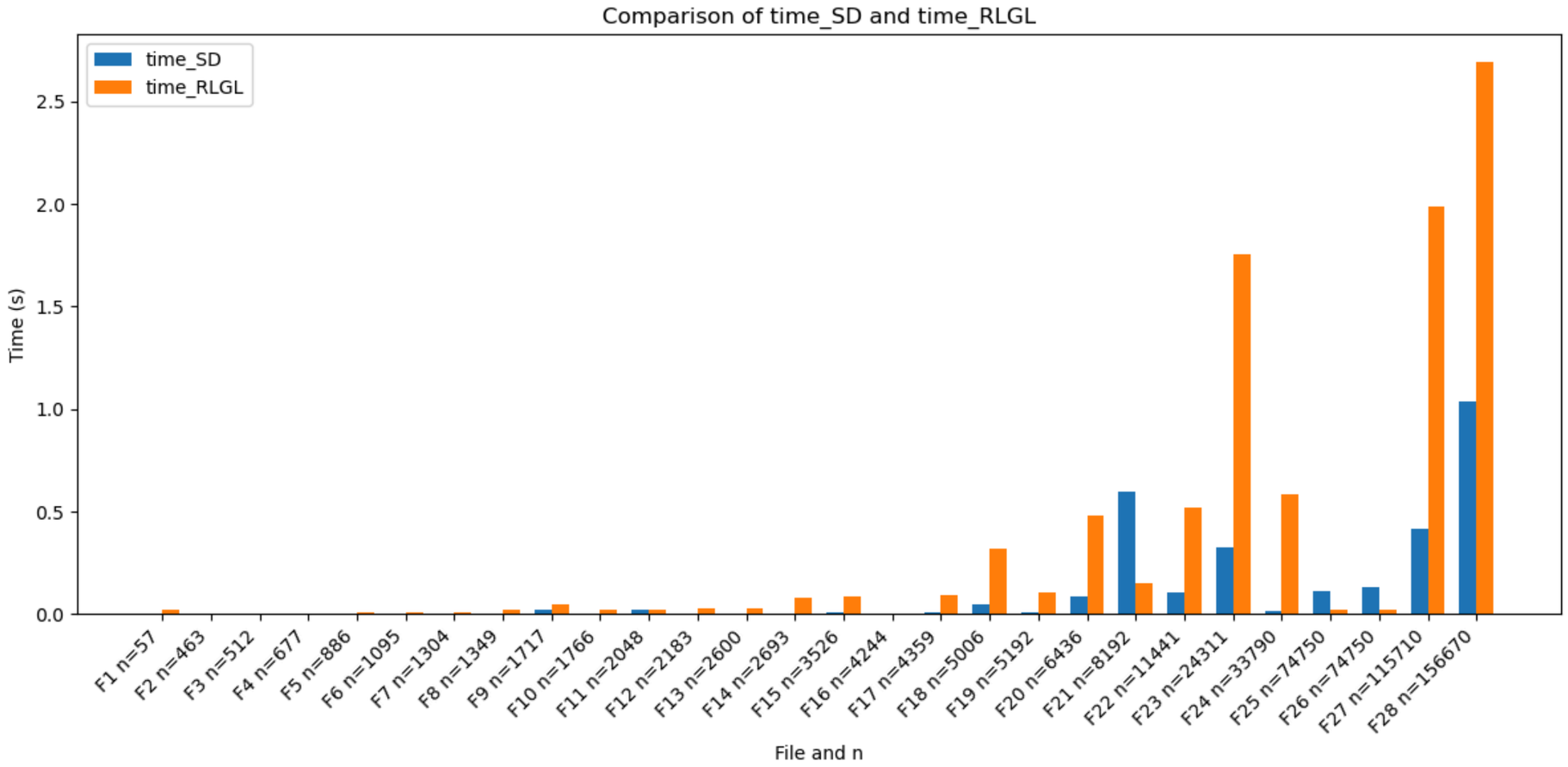
Deux catégories de fichiers:

- Fichiers avec paramètres codés en dur
- Exemple :
  - csma.2-2.jani (N = 2, K = 2)
    - 10 actions
    - 1,038 states
  - csma.2-4.jani (N = 2, K = 4)
    - 10 actions
    - 7,958 states

- Fichiers paramétrables :
- Exemple :
  - consensus.2.jani (N = 2)
    - K = 2
      - 3 actions
      - 272 states
    - K = 4
      - 3 actions
      - 528 states

- Notre objectif: comparer la performance de plusieurs algorithmes

Temps d'exécution avec des modèles aux paramètres fixes



Nous pouvons rien conclure.

Problème: le temps d'exécution dépend fortement du modèle

→ utiliser un même modèle et faire varier la taille (n)

Fichiers paramétrables sont donc préférés pour nos tests

Description du test

Méthodologie de test:

- varier un paramètre lié à la taille de l'espace d'états
- exécuter chaque fonction 30 fois pour chaque instance
- données sauvegarder dans des fichiers .csv

Algorithme à tester:

- MC:
  - StationaryDistribution
  - StationaryDistributionRLGL
- MDP
  - ValueIteration
  - ValueIterationGS
  - PolicyIterationModified
  - PolicyIterationModifiedGS
  - RelativeValueIteration (Average MDP)

Classification et sélection des modèles

Nombre de models réalisables pour DTMC: 8

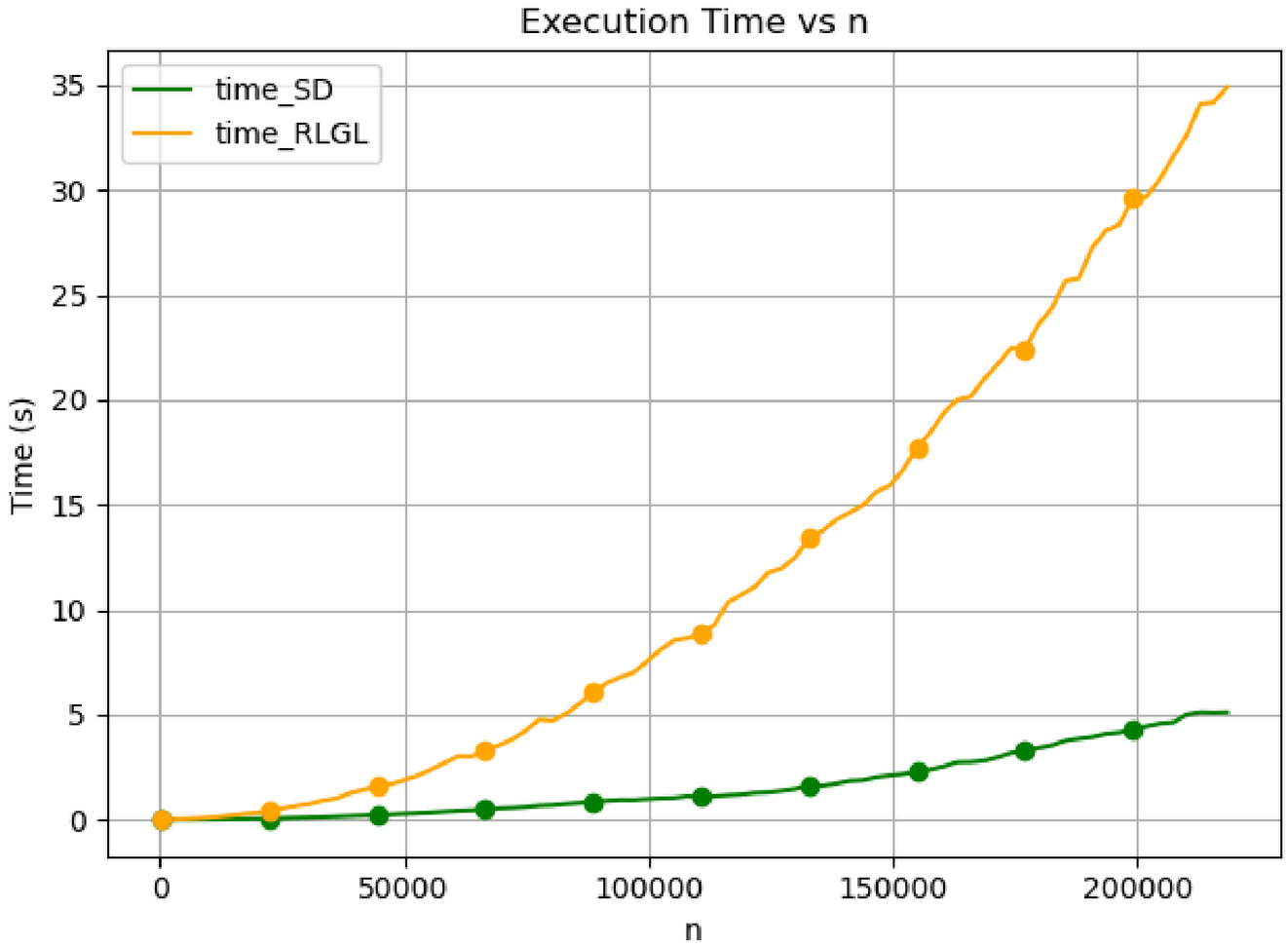
- dont 4 codés en dur et 4 paramétrables
- Modèle représentatif choisit : brp.jani

Nombre de models réalisables pour MDP: 27

- dont 16 codés en dur et 11 paramétrables
- Modèles représentatifs choisits: consensus.2.jani et firewire_dl.jani

Temps d'exécution en fonction de nombre d'états

Fichier de test: brp.jani

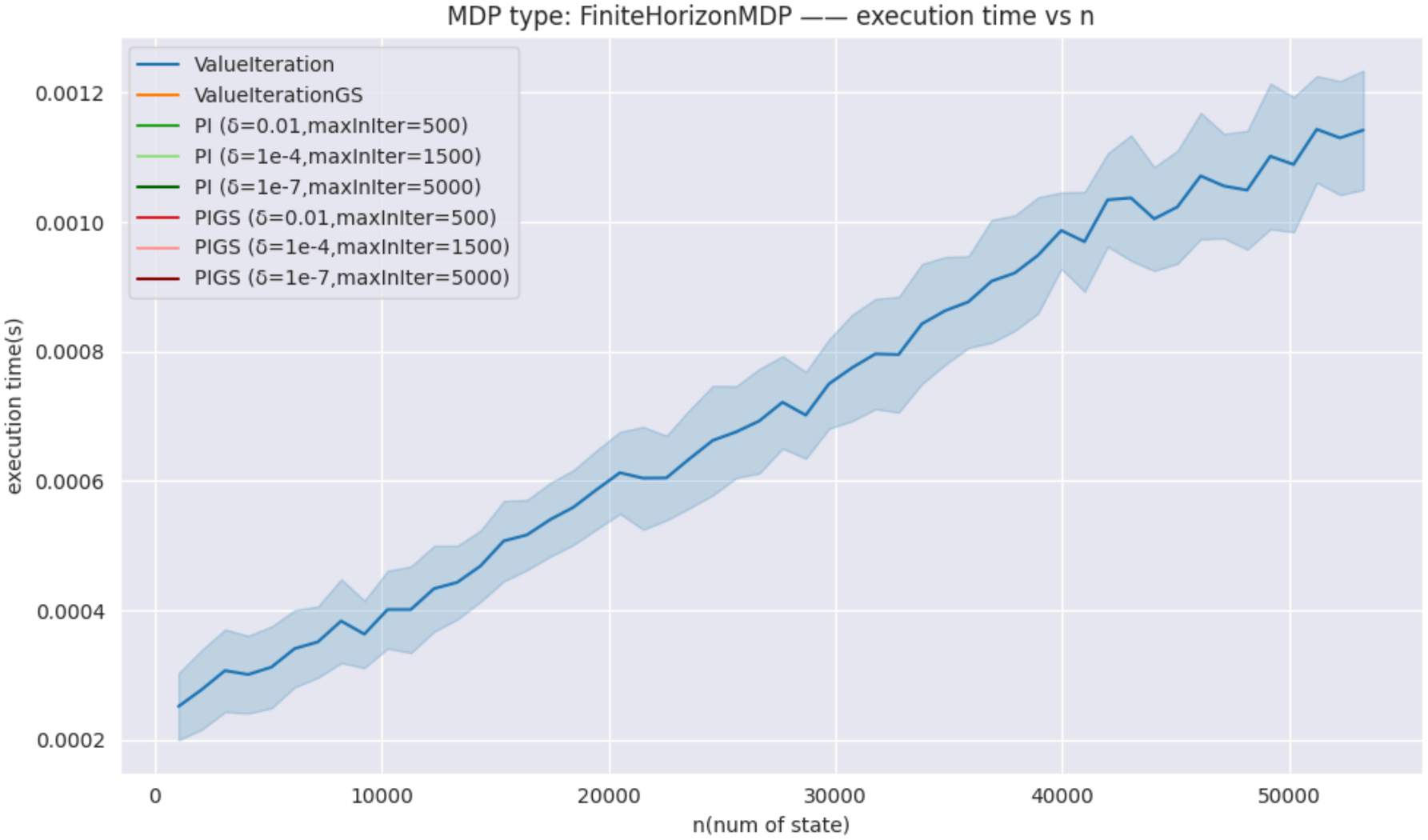


- Résultat direct du graphe:
  - SD plus rapide que SD_RLGL
- Avec *log* en terminal:
  - petit *n*, SD 10 fois plus d'itérations que RLGL
  - Leur limite:
    - SD: 2000
    - RLGL: 1000
- Avec les sorties:
  - deux distributions initiales différentes:
    - SD: concentrée sur l'état 0
    - RLGL: uniforme sur tous les états

(Possible de vérifier les propriétés de MC dans Marmote)

Temps d'exécution en fonction de nombre d’états avec critère horizon finite

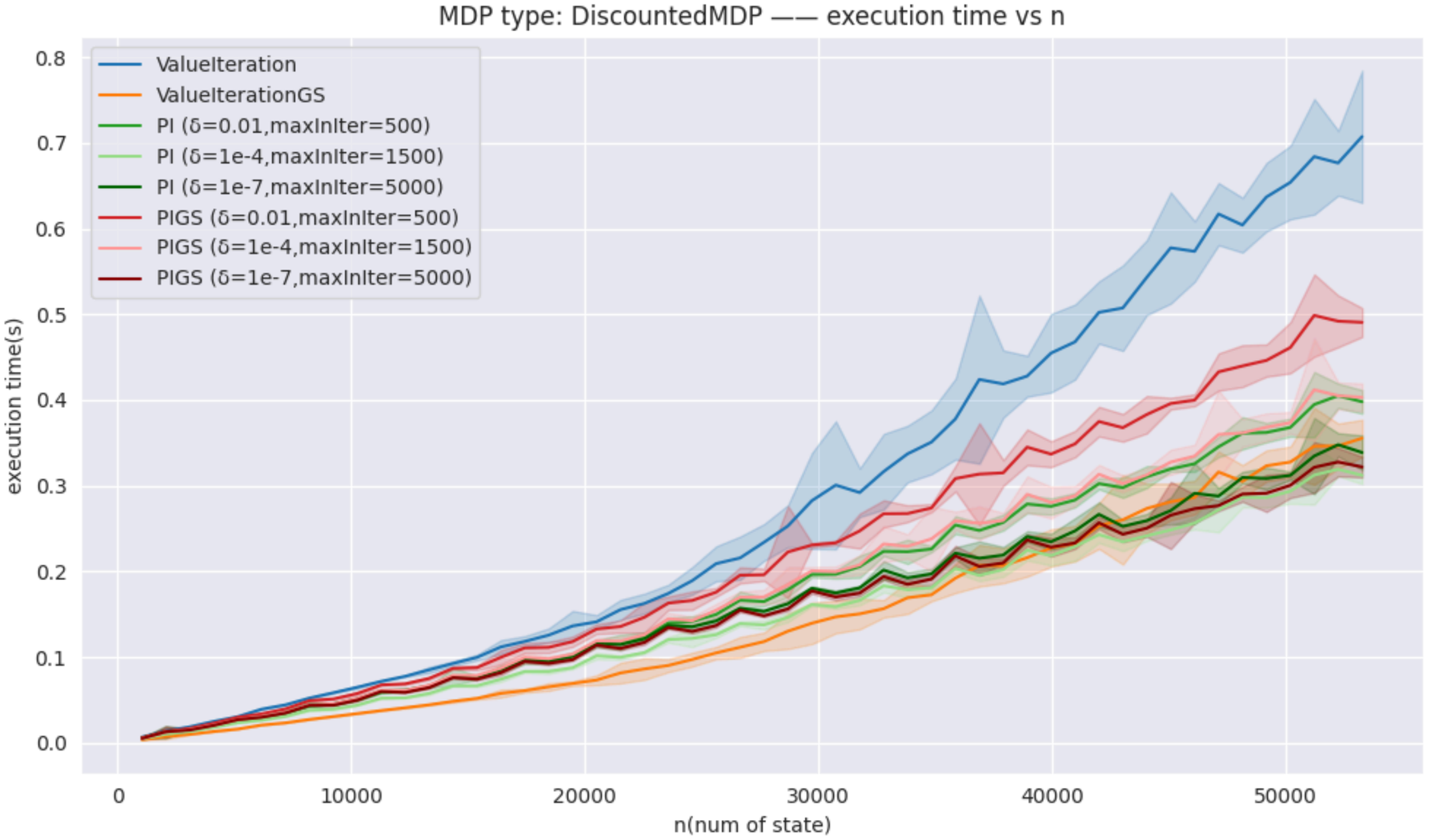
Fichier de test: consensus.2.jani. Itération maximale = 5000



- Complexité pseudo-linéaire
- Résolvable par une programmation dynamique classique

Temps d'exécution en fonction de nombre d'états avec critère Discounted

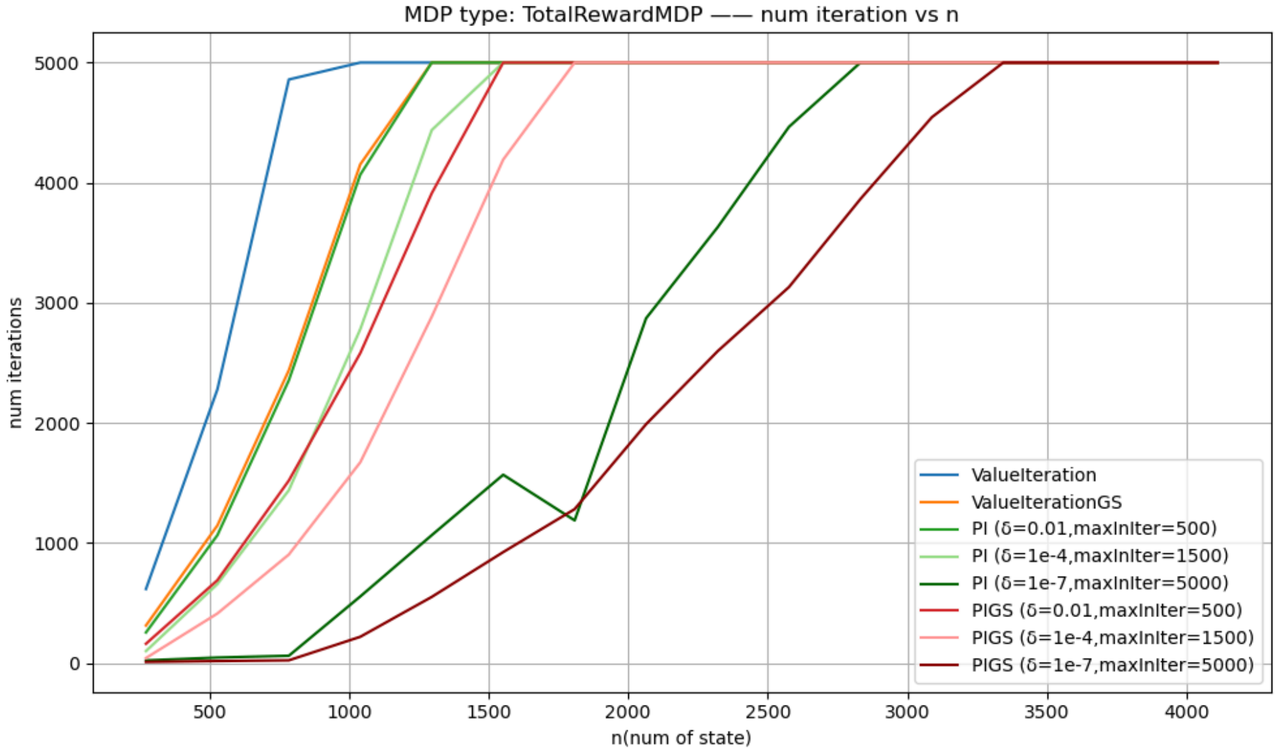
Fichier de test: consensus.2.jani. Itération maximale = 5000



- Toutes les méthodes convergent
- VI est lente et présente une variabilité un peu élevée
- Les méthodes PIM-GS sont moins performantes que les méthodes PIM classiques

Temps d'exécution en fonction de nombre d'états avec critère total

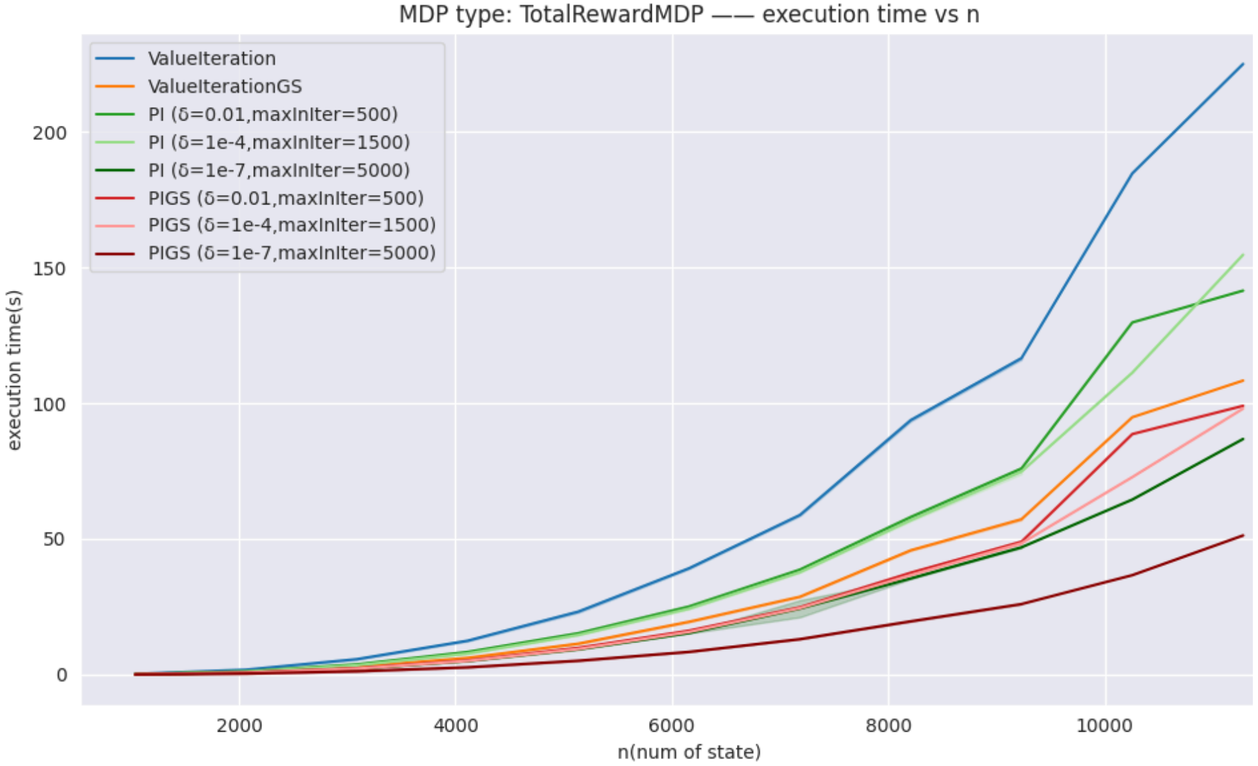
Fichier de test: consensus.2.jani. Itération maximale = 5000

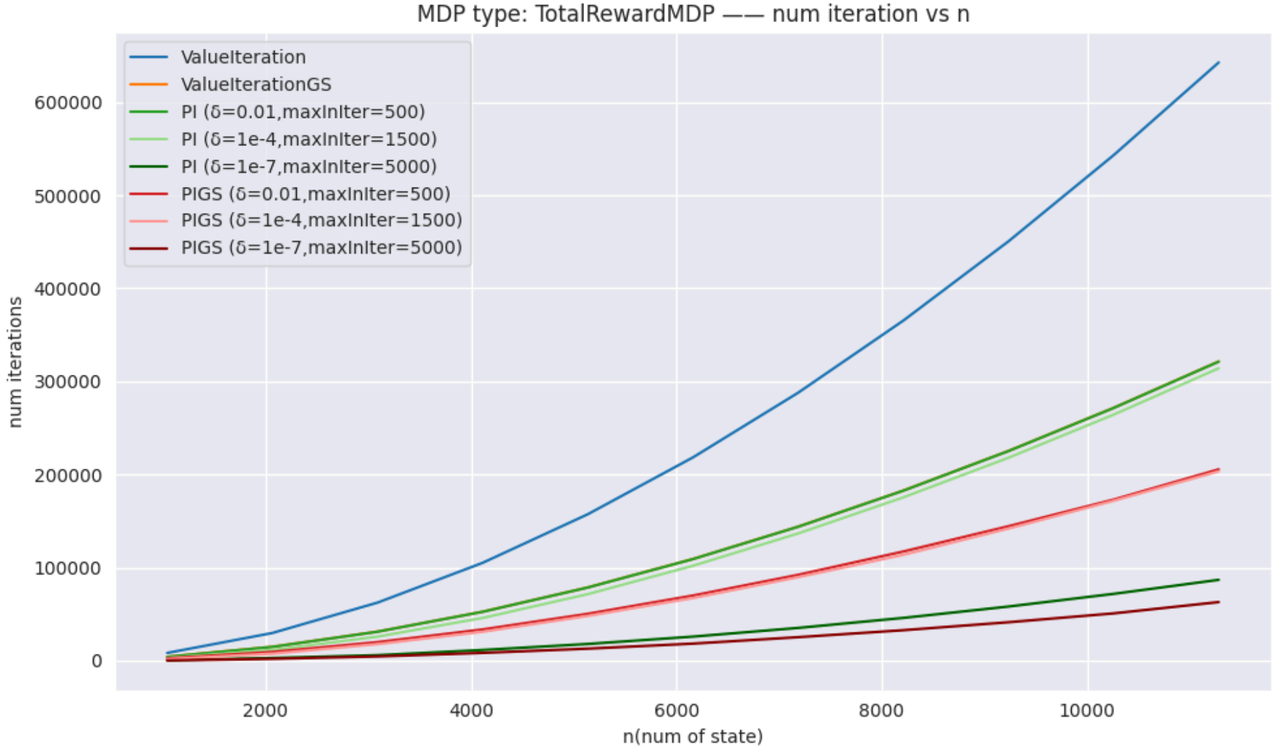


- Aucune méthode n'a convergé
- Le critère total nécessite un nombre d'itérations plus élevé que le critère γ-pondéré

Temps d'exécution en fonction de nombre d'états avec critère total

Fichier de test: consensus.2.jani. Itération maximale illimitée (1 miliards)

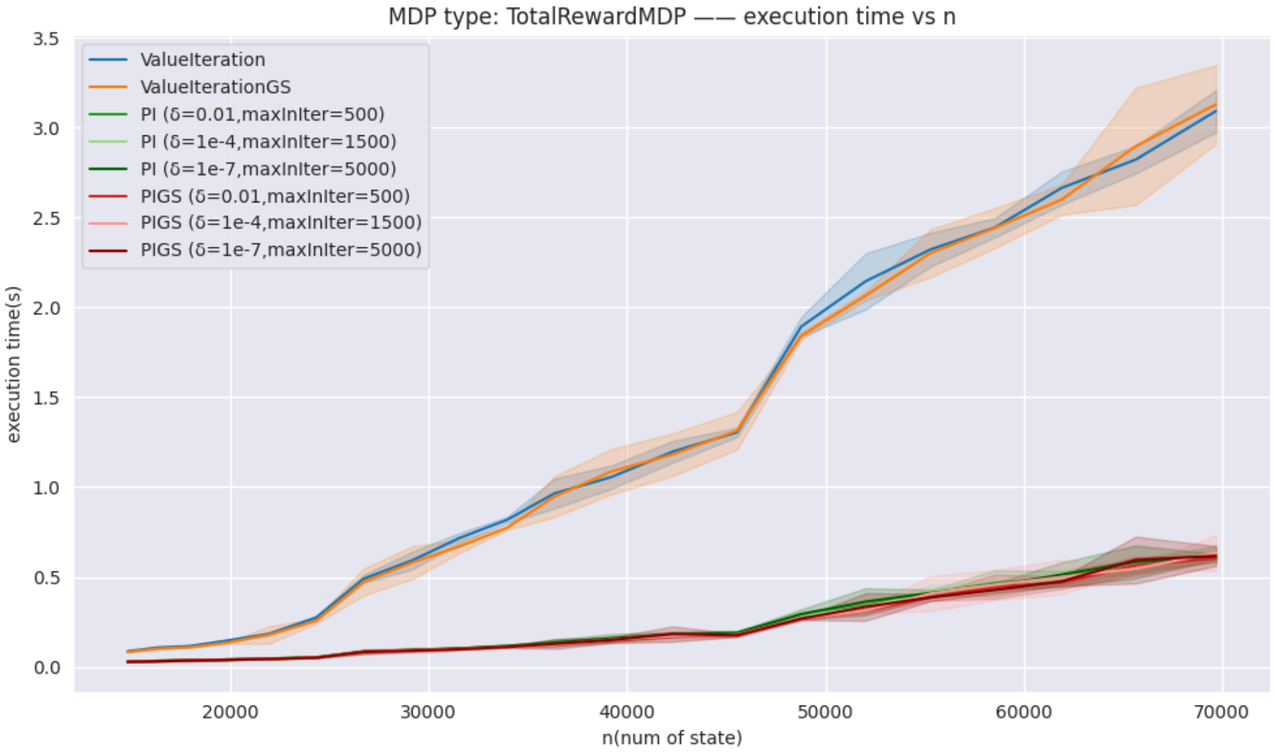


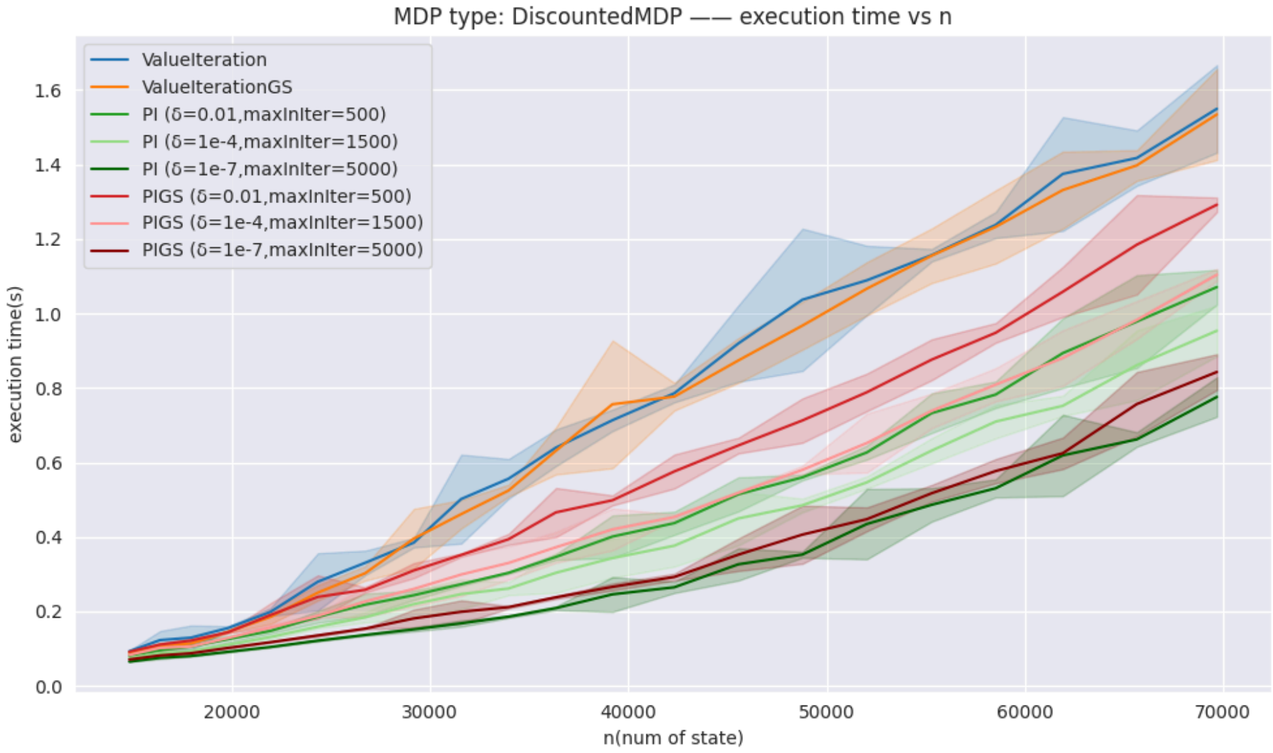


- Toutes les méthodes convergent
- Temps de calcul élevé
- Les méthodes PIM sont plus rapides que les méthodes VI

Temps d'exécution en fonction de nombre d’états

Fichier de test: firewire_dl.jani

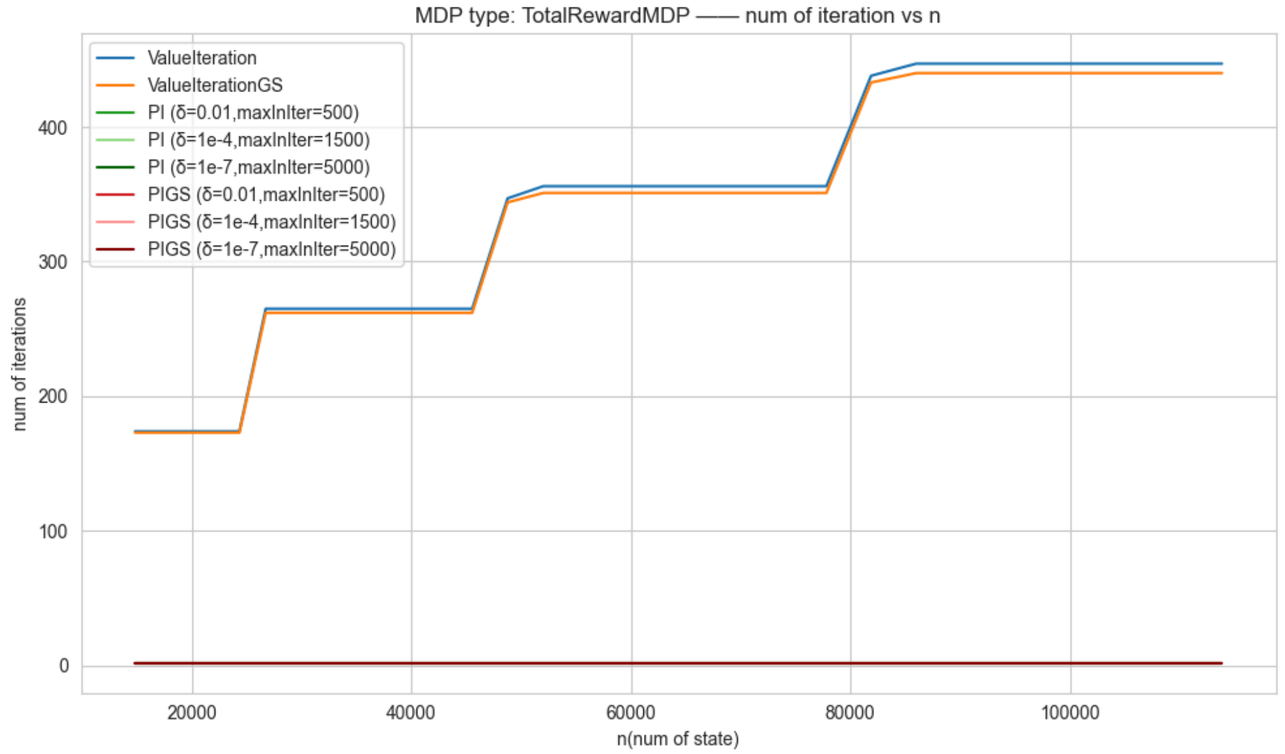


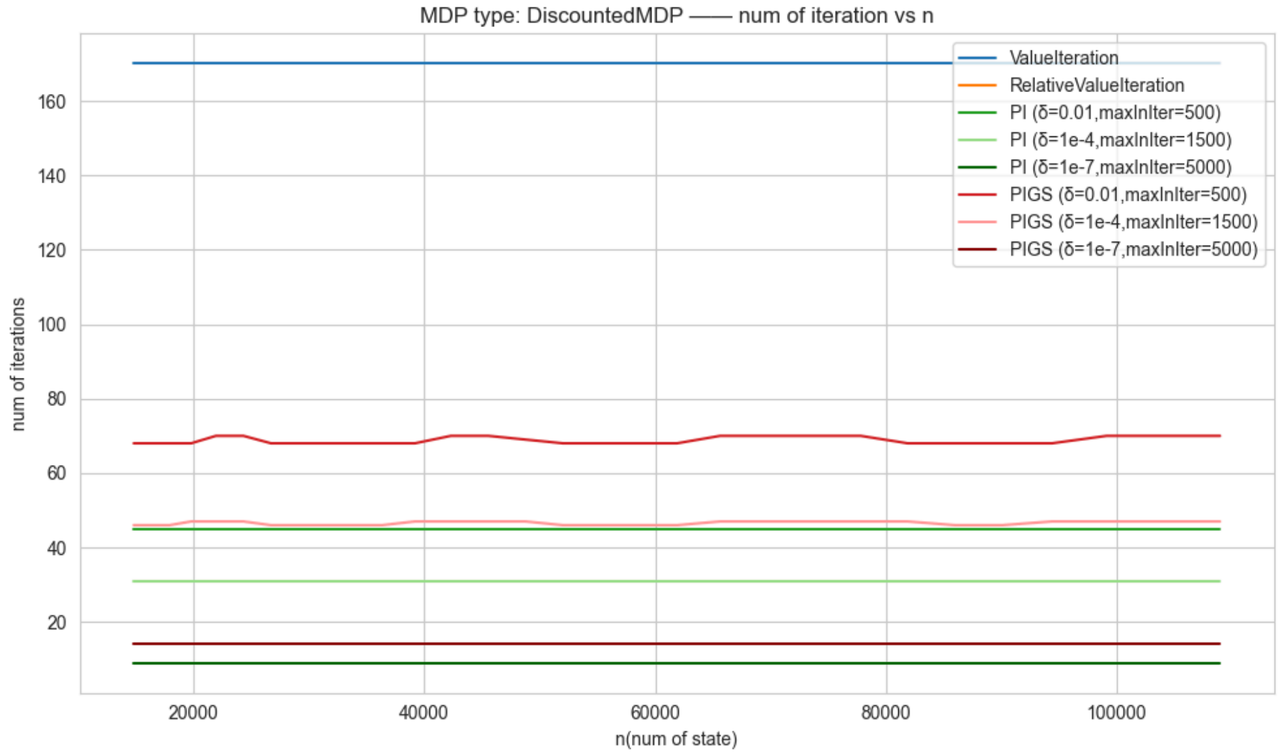


- Toutes les méthodes associées à ces deux critères convergent
- PIM est plus rapide que VI pour les deux critères
- Les méthodes PIM-GS sont toujours moins performantes que les méthodes PIM classiques

Nombre d'itération en fonction de nombre d'états

fichier de test: firewire_dl.jani





- Les méthodes PIM convergent instantanément (nombre d'itérations = 2)
- Augmentation non-linéaire (en forme d'escalier) du nombre d'itérations pour les méthodes VI

Observations sur les limites rencontrées

JANI

- centrés sur les problèmes de reachability
- certains fichiers ne permettent pas de varier le nombre d’états.
- benchmarks orientés model checking

Marmote

- MarmoteBox génère tous les états possibles et il existe bug
- certains logs difficiles à récupérer
- manque de getters

Conclusion

- Ce que nous avons fait
  - Implémentation d'un parser JANI → Marmote
  - JANI-R comme une extension de JANI
  - Réalisation de benchmarks systématiques sur l'ensemble d'algorithmes dans Marmote

- Améliorations possibles:
  - Trouver (ou créer) des instances de MDP plus générales (autres problèmes que le reachability)
  - Intégration d'autres solveurs (ex: MDPToolbox) pour faire de comparaison
  - Extension du parser à d'autres langages (ex: RDDL)

Remerciement