Topological Planning with Post-Unique and Unary Actions

Intervenant : **Guillaume Prévost**

Conférence : France@Internationnal (IJCAI – 23)

Date : 05 juillet 2023

Lieu : Strasbourg (France)

Auteurs:

- **Guillaume Prévost**, **Stéphane Cardon**. Académie Militaire de Saint-Cyr Coëtquidan, CReC Saint-Cyr
- **Tristan Cazenave**, Université Paris Dauphine – PSL, LAMSADE, CNRS
- **Christophe Guettier**, Safran Electronics & Defense
- **Éric Jacopin**, Hawkswell Studios

De quoi parle notre papier

- ✓ Planification d'actions, avec SAS.
- ✓ Appliquée aux Personnages Non-Joueurs (PNJs) dans les jeux vidéos.
  - Planificateurs frugaux. But : réduire la charge de calculs du processeur.
  - Contrainte de temps : 1,67ms – 10% du temps entre 2 images à 60 FPS.
- ✓ Nouvelles classes de problèmes : SAS-PUC_k
  - SAS-PUC_0, SAS-PUC_2^S, SAS-PUC_2^* sont informatiquement traitables.
- ✓ Nouvel algorythme : *TopoPlan*
  - Correct, complet avec une complexité temporelle linéaire.
  - Rapide : 3.400.000 plans avec au plus 10 d'actions en temps réel.

Etudes préliminaires

Goal-Oriented
Action Planning
GOAP

Objectif : Fournir un plan au plus de PNJs possibles en temps réel (1,67ms).

Actuellement :

- Planificateur basé sur A*.
- Problèmes de planification (PNJs) simples.
  - Actions unaires
- Astuces pour contrer surcharge du processeur :
  - Elagage contextuelle, i.e. actions contextuellement post-uniques
  - Plans courts (< 5 actions)
  - Limite d'appels au planificateur
  - Etc.

Hypothèses de travail

1. Les actions sont **Post-uniques**.
2. Les actions sont **Unaires**.
3. Les plans sont **Totalement ordonnés**.
4. Il **n'y a pas deux fois** la même action dans le plan solution.

Comment faire mieux ?

Réduire la complexité algorithmique du planificateur

- GOAP utilise A* : complexité exponentielle [7].
- La planification d'actions est difficile par nature [6].

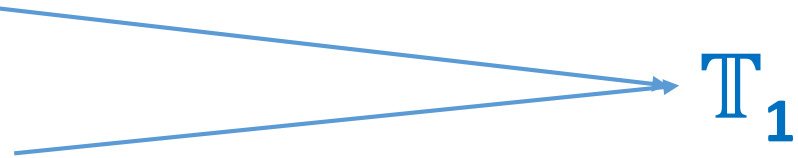
- Choix d'un formalisme de planification :
  - *Simplified Action Structure* (SAS) [5]

- Imposer des règles, appelé *restrictions*, dans le formalisme choisi :
  - (P) Post-Unique
  - (U) Unarité

SAS-PU est une classe de problèmes **non-traitable** informatiquement [5].

SAS-PU$\mathbb{T}_1$ [10]

- SAS-PU est non-traitable informatiquement car il existe des plans solutions minimales d'une taille exponentielle en fonction de la taille du problème (ex : Gray Code).
- En d'autres termes : certaines actions apparaissent plusieurs fois dans un plan solution minimal.
- Dans les jeux vidéos étudiés, aucun plan n'a deux fois la même action. Où, chaque action apparaît au plus qu'une seule fois : $\mathbf{T_1}$
- Les plans sont totalement ordonnés : $\mathbf{T}$



T_1 : Il n'y a pas 2 fois la même action

- T_1 est une restriction de sortie.

- Dans notre papier IJCAI, on propose 3 restrictions d'entrée créant des sous-classes SAS-PU dont les plans solutions sont T_1 :
  - SAS-PUC_0
  - SAS-PUC_2^S
  - SAS-PUC_2^*

L’Eleveur de chevaux [1,2,9]

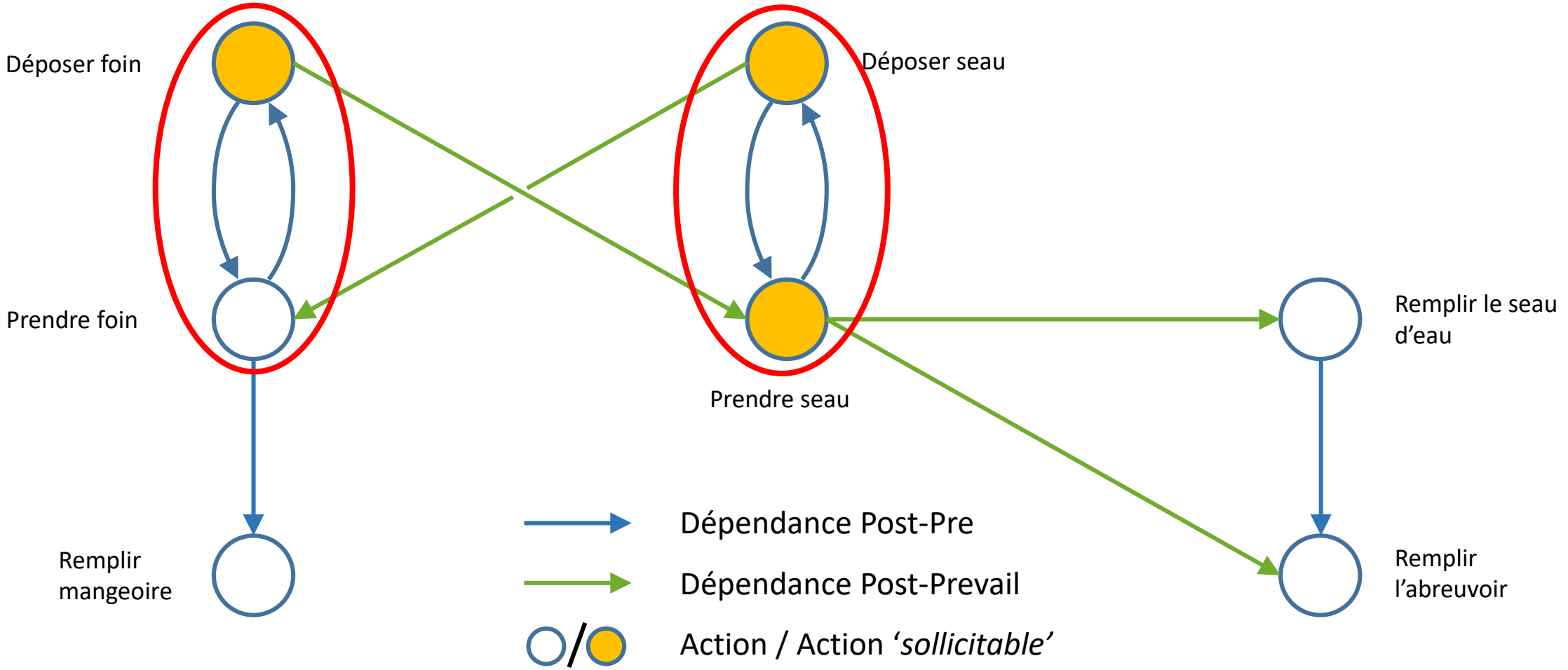
Exemple d’un problème SAS-PU

Début du travail : l’éleveur de chevaux ne porte rien et doit remplir l’abreuvoir d’eau et le mangeoire de foins.

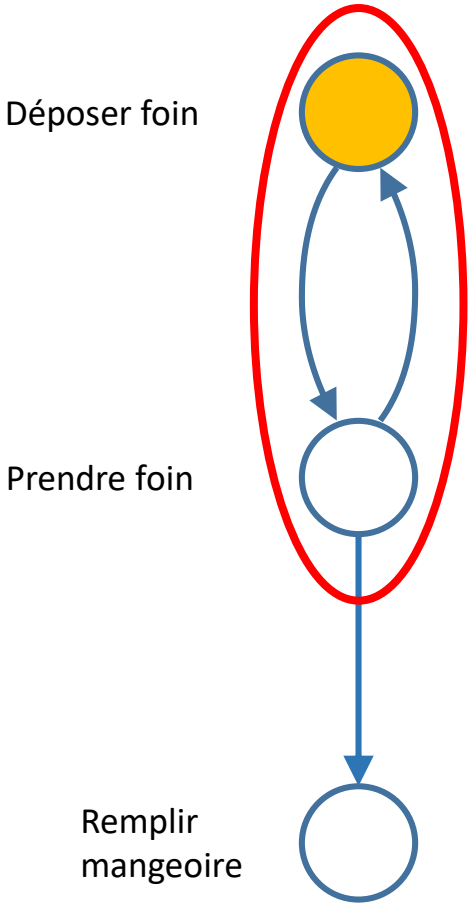
Fin de journée : L’éleveur de chevaux a rempli le mangeoire et l’abreuvoir. Le seau d’eau a été reposé.

L'Eleveur de chevaux (Graphe d'actions)

Exemple d'un problème SAS-PU



Lemme et restriction C_k



Lemme :

Pour un problème SAS-PU, un graphe d'actions de domaine est soit un **arbre dirigé**, soit un **graphe dirigé avec un cycle unique**.

Restriction C_k:

On dénote C_k la restriction structurelle qui limite à au plus $k \in \mathbb{N}$ le nombre d'actions dans l'unique cycle et qui possède au moins une action orange (i.e. *sollicitable*).

SAS-PUC$_k$ avec $k \in \mathbb{N}$

$k = 0$

Théorème :

SAS-PUC_0 ont des plans solutions sans doublon, i.e. **T$_1$**.

$k = 1$

SAS-PUC_1 n'a pas de sens car : post-condition ≠ pre-condition (restriction S4 [CB, thesis 1992]).

$k = 2$

Théorème :

Les problèmes SAS-PUC_2 sont généralement non-traitables informatiquement.

$k \geq 3$

Lemme :

Pour $k \geq 3$, il existe des instances de problème SAS-PUC_k où le plan solution minimal est au moins **T$_2$**.

SAS-PUC_2^S / SAS-PUC_2^*

SAS-PUC_2^S :

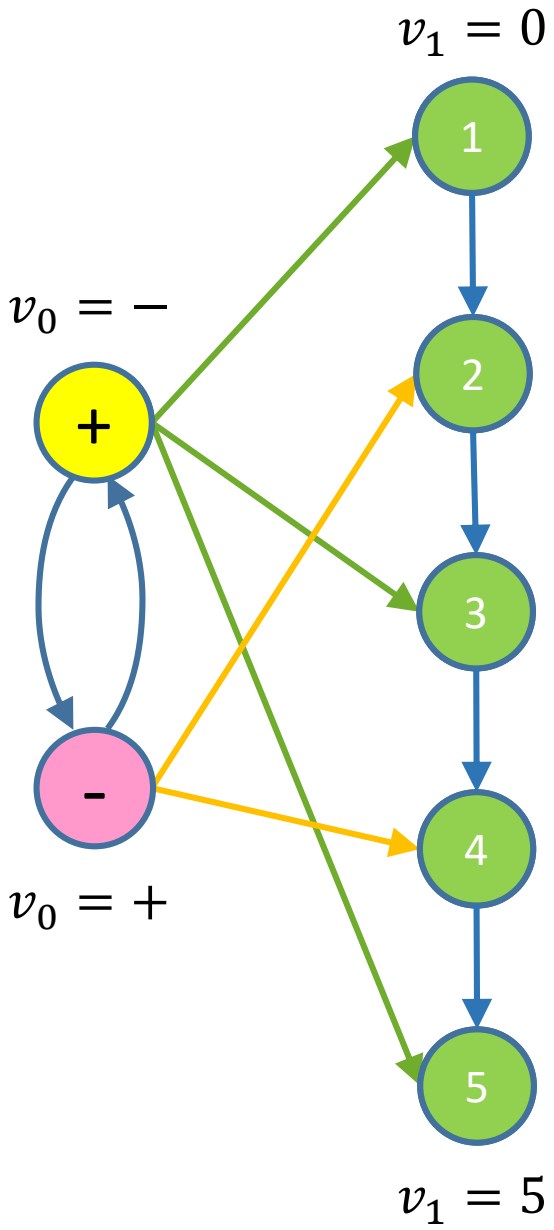
SAS-PUC_2^S est une sous-classe de SAS-PUC_2 et il y a au plus une action « *sollicitable* » par cycle.

SAS-PUC_2^* :

SAS-PUC_2^* est une sous-classe de SAS-PUC_2 et pour tout cycle ayant deux actions « *sollicitable* », dénommées *On* et *Off*, alors les actions nécessitant *On* ne doivent pas être reliées aux actions nécessitant *Off* dans le graphe d'actions privé du cycle {*On, Off*}.

Théorème :

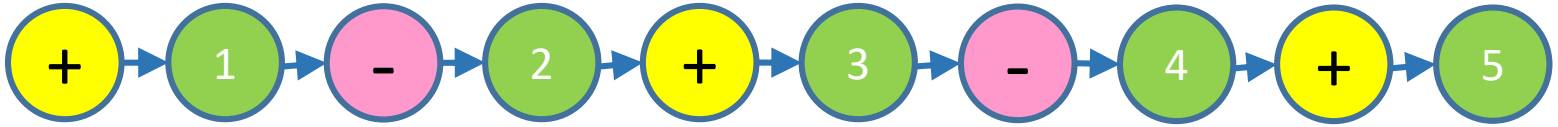
Les problèmes SAS-PUC_2^S et SAS-PUC_2^* ont des plans solutions sans doublon, i.e. $\mathbf{T_1}$.



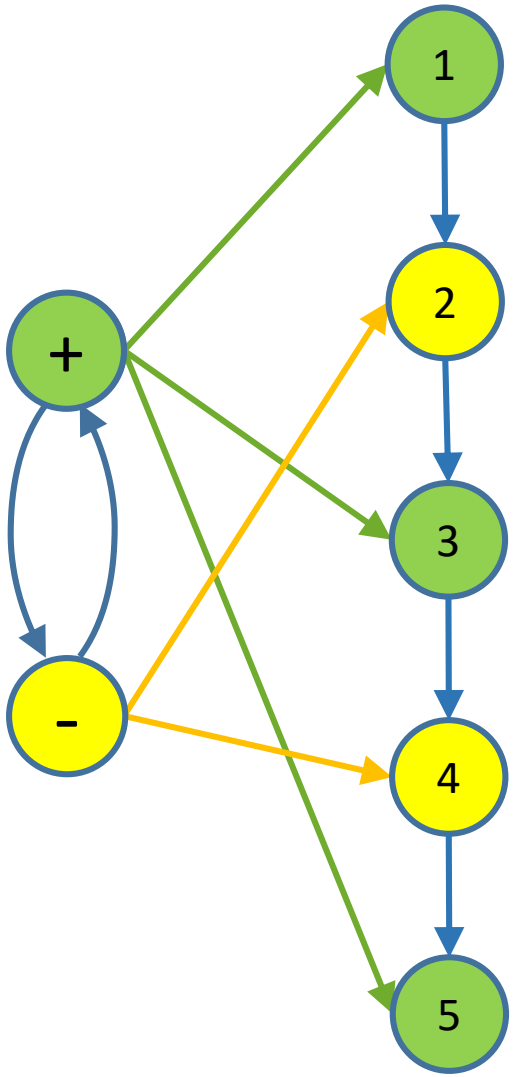
SAS-PUC_2

$$Start = \langle -, 0 \rangle ; Goal = \langle +, 5 \rangle$$

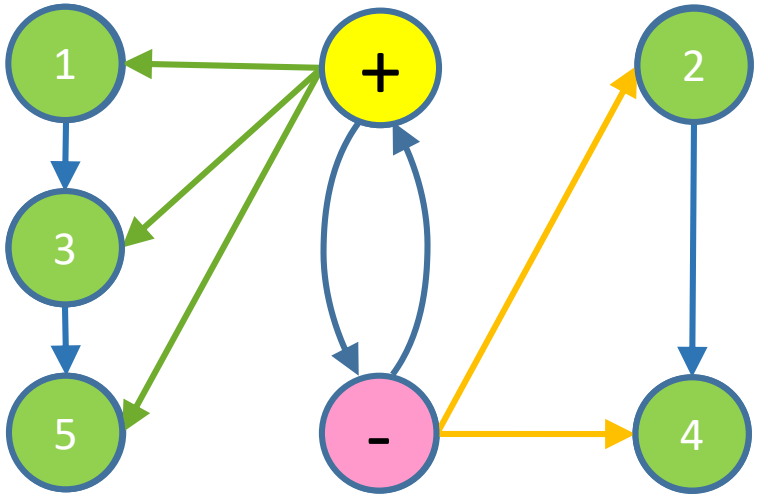
Plan solution minimal :



SAS-PUC_2

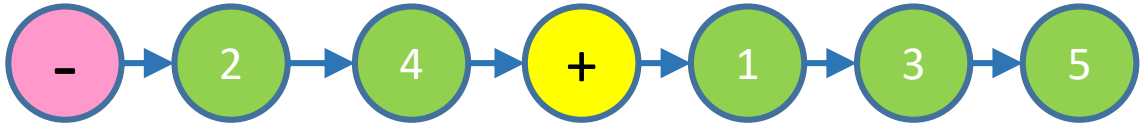


SAS-PUC_2^*

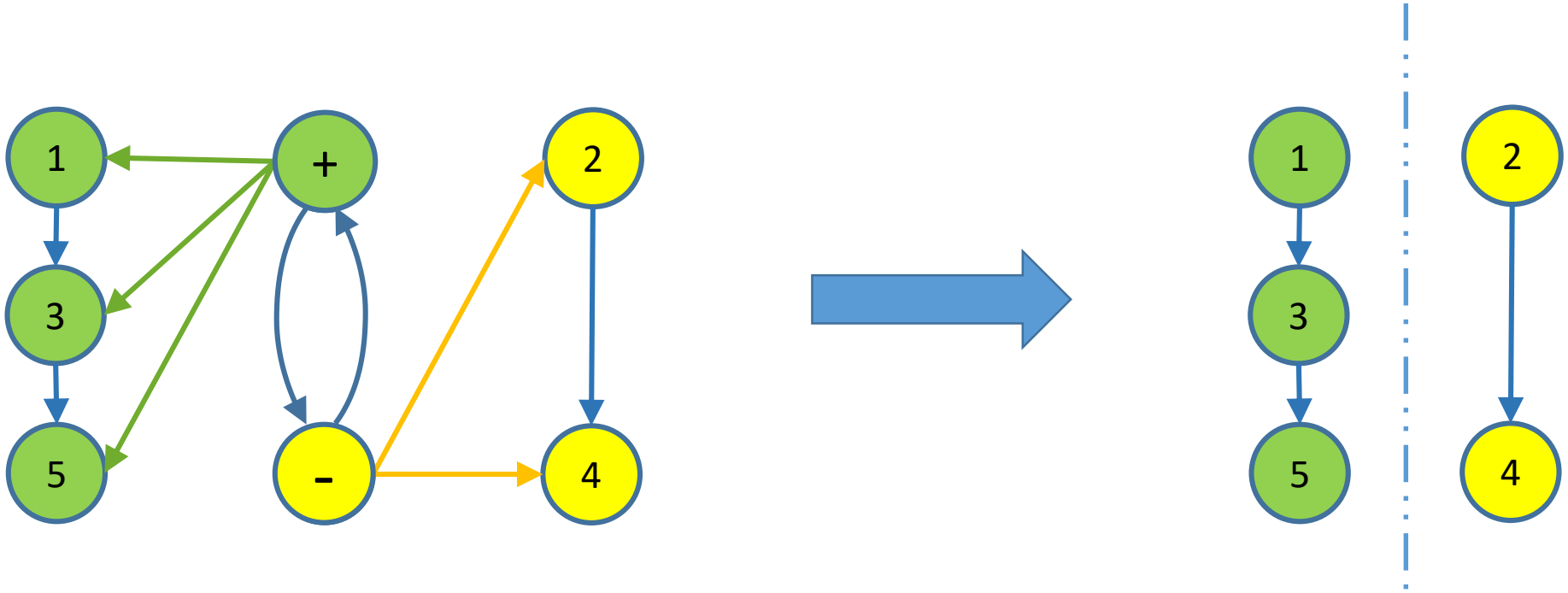


$$\text{Start} = \langle +, 0{,}0 \rangle \; ; \; Goal = \langle +, 4{,}5 \rangle$$

Plan solution minimal :



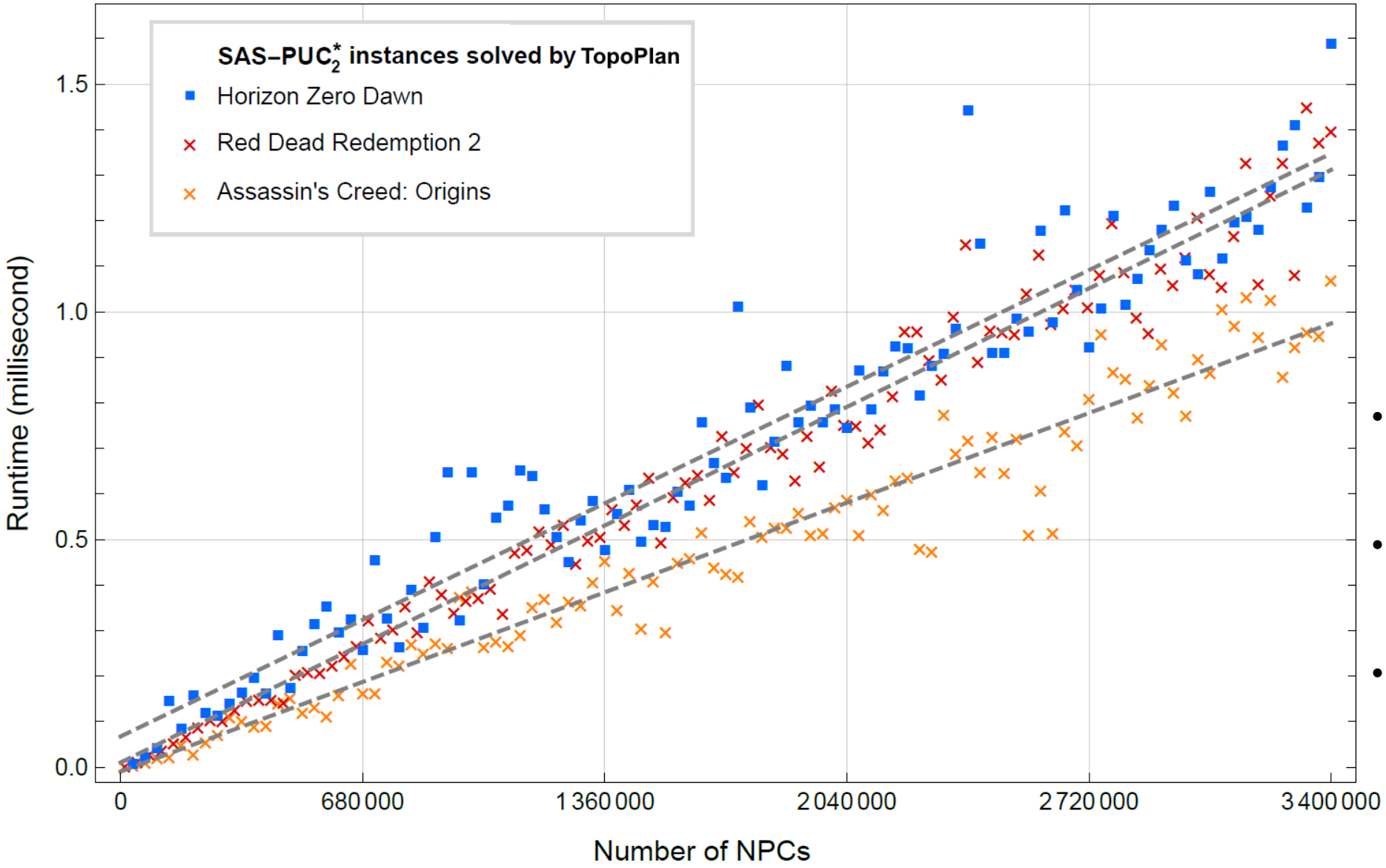
SAS-PUC_2^*



TopoPlan

TopoPlan :

- **Input**: $\prod$, l'instance d'un problème **SAS-PUC_0**, **SAS-PUC_2^S** ou **SAS-PUC_2^***.
- **Output**: Si $\prod$ peut être résolue, alors *TopoPlan* retourne un plan solution de $\prod$, minimal, linéaire et sans doublon d'action. Si $\prod$ ne peut être résolue, alors *TopoPlan* indique un échec.
- *TopoPlan* est correct et complet.
- Complexité temporelle : $O(A + E_A)$, avec A l'ensemble des actions, E_A l'ensemble des relations entre actions.
- Complexité spatiale : $O(A^2)$



- Rapide : 3,4 millions de plans en 1,5ms (< 1,67ms).
- Plans contenant entre 2 et 10 actions.
- Références [1,2]

TopoPlan codé avec C++14 et les paramètres par défaut de VS2019. Expériences menées avec un AMD Ryzen 7 2700X, 32Gb of RAM, Windows 10 (64 bits).

Discussion

- Réduction de la complexité algorithmique.
- Classes permettant de modéliser des problèmes de planification concrets.
    - (P) reste parfois non-trivial à respecter.
- Résultats surprenants
    - Codage C/C++, allocation mémoire, utilisation de pointeurs…
    - Logiciel AMD « µProf » : utilisé pour vérifier le travail du processeur.
- *Low Tech* :
    - *AMD Ryzen™ 7 2700X*
    - Consommation < 100 Watts

Perspectives

- Planificateur avec une complexité logarithmique ?
- Paralléliser et utiliser les GPU [8] ?
- Quels résultats sur d'autres processeurs ?

Références :

1. Guillaume Prévost. *Real-time Planning: Reducing Complexity for Scaling Up*. Thèse de doctorat, LAMSADE, Université Paris-Dauphine PSL, Décembre 2022.
2. Guillaume Prévost, Stéphane Cardon, Eric Jacopin, Tristan Cazenave, and Christophe Guettier. *Planning for millions of NPCs in real-time*. 2022 IEEE Symposium Series on Computational Intelligence (SSCI), pages 330–336. IEEE, 2022.
3. Guillaume Prévost, Stéphane Cardon, Eric Jacopin, and Tristan Cazenave. *Planification d'actions dans les jeux-vidéos : À quoi servent les coûts des actions*. Rencontres des Jeunes Chercheurs en Intelligence Artificielle (RJCIA), Toulouse, 2019.
4. Peter Higley. *GOAP from FEAR to Shadow of Mordor*, Monolith Productions, https://youtu.be/gm7K68663rA. Game Developers Conference (GDC), March 2015.
5. Christer Bäckström. *Computational Complexity of Reasoning about Plans*. PhD thesis, Department of Computer and Information Science, Linköping University, September 1992.
6. David Chapman. *Planning for conjunctive goals*. Artificial intelligence, 32(3):333–377, 1987.

Références :

7. Peter E Hart, Nils J Nilsson, and Bertram Raphael. *A formal basis for the heuristic determination of minimum cost paths*. IEEE transactions on Systems Science and Cybernetics, 4(2):100–107, 1968.
8. Stéphane Cardon et Éric Jacopin. *Binary GPU-planning for thousands of NPCs*. IEEE Conference on Games (CoG). IEEE, 2020. p. 678-681.
9. DefendTheHouse. *NPC daily life in Read Dead Redemption 2*. https://youtu.be/MrUJJgppMn4?t=434, November 2018.
10. Guillaume Prévost, Stéphane Cardon, Eric Jacopin, and Tristan Cazenave. *La planification SAS sous forme de tri topologique*. Conférence Nationale en Intelligence Artificielle (CNIA), Saint-Étienne, 2022.