Apprentissage par renforcement pour la synthèse de contrôleur logique et implémentation en langage ST

Dimitri Renard, Ramla Saddem, David Annebicque, Bernard Riera

Société d'Automatique,
de Génie Industriel & de Productique

Apprentissage par renforcement pour la synthèse de contrôleur logique et implémentation en langage ST

Dimitri Renard[1,2], Ramla Saddem[1], David Annebicque[1], Bernard Riera[1]
[1] CReSTIC, Université de Reims Champagne-Ardenne, Reims, France
prenom.nom@univ-reims.fr
[2] Prosyst, Valenciennes, France
drenard@prosyst.fr

Mots-clés *: Apprentissage automatique, Commande, IEC 61131-3, API, SED.*

1 Introduction

L'industrie manufacturière est dans une optique de décarbonation. Pour cela, elle doit produire avec plus de qualité, de flexibilité mais aussi moins d'énergie tout en gardant les cadences de production. Les technologies de l'industrie 4.0 apportent de nouveaux outils couvrant les données des ateliers jusqu'à leurs analyses. Cependant, l'ajout de ces outils doit être corrélé avec l'évolution des méthodes et pratiques existantes. Les méthodologies de conception de la commande en font partie, d'autant plus que la création d'un contrôleur logique flexible sur un système complexe est une tâche difficile et par extension couteuse. La démocratisation des algorithmes d'intelligence artificielle (IA) tend à faciliter la réalisation de tâches complexes. Le but de ce résumé est de présenter une méthodologie de génération d'une commande pour contrôleur logique générée par IA jusqu'à son implémentation en langage IEC 61131-3 dans un Automate Programmable Industriel (API).

2 Commande pour contrôleur logique générée par IA

2.1 Etat de l'art

L'apprentissage par renforcement est une des catégories de l'IA. Il se base sur le principe de l'essai-erreur afin de trouver l'action à réaliser pour accomplir au mieux ses objectifs. Pour cela, le modèle appelé agent reçoit l'état de son environnement en entrée. Il choisit une action à réaliser sur l'environnement. En fonction du résultat de l'action sur l'environnement, un gain plus ou moins important est retourné à l'agent. Au fur et à mesure des essais, l'agent apprend à choisir l'action la plus profitable dans chaque état [5]. Les travaux sur la génération d'une commande de contrôleur logique via la puissance des algorithmes d'IA sont encore peu nombreux. Les travaux de F. Jaensch utilisent le *virtual commissionning* pour réaliser l'apprentissage de leur système [1,2]. Ces travaux

insistent sur une décomposition des systèmes afin de réduire la complexité d'apprentissage de chaque sous-système. Les travaux de J. Zinn, montre une génération du contrôleur dans un cas d'usage "simplifié" avec une abstraction des actions pour l'agent, afin de réduire l'explosion combinatoire [6]. Une analyse de la littérature des approches existantes en IA pour la planification et le contrôle des Systèmes à Evènements Discrets révèle un champ de recherche prometteur [3]. Cependant, aucun de ces travaux ne présente un état généralisable ainsi qu'une implémentation concrète d'un code en langage IEC 61131-3.

2.2 Apprentissage par renforcement pour la synthèse de contrôleur logique

Une des principales problématiques lors de l'application d'algorithmes d'apprentissage par renforcement est l'explosion combinatoire. Ainsi, pour obtenir la commande de contrôleurs logiques dans un temps acceptable pour un automaticien, il est nécessaire de contrôler cette explosion combinatoire. La définition des états et des actions est donc contrainte par cet objectif. L'état du système permet de se ramener à un problème combinatoire. Afin de réduire l'espace d'états, celui-ci est construit à partir de tâches opératives du système. Une tâche opérative dispose d'une condition initiale (CI), un état initial (IS). Elle ne peut être interrompue. Le vecteur d'état du système se décrit sous la forme générique [CI1, IS1, CI2, IS2, …]. L'espace d'actions est aussi une cause d'explosion combinatoire. Les actions ne sont pas directement les activations d'actionneurs mais les autorisations de tâche opérative. De plus, comme proposé par J. Zinn [6], nous n'autorisons qu'une tâche par action. Cependant, au déclenchement d'une tâche, l'état est modifié au cycle API suivant et une nouvelle tâche peut être déclenchée. Afin de limiter encore plus l'espace d'états, un filtre logique [4] est ajouté en amont du choix de la tâche lors de l'apprentissage. Il permet d'ajouter des contraintes connues entre des tâches ne pouvant s'exécuter parallèlement. L'environnement est réalisé en simulation de partie opérative (PO) afin d'accélérer le temps et parer à toutes erreurs de contraintes entre tâches. Le résultat de l'apprentissage est une table état/valeur des actions (E/V), représentant le gain espéré après l'autorisation d'une tâche dans un état donné.

2.3 Implémentation dans un API

Afin d'implémenter le modèle dans un API, nous devons simplifier au maximum les traitements effectués sur celui-ci. La première étape consiste à transformer la table E/V en une table état/action en sélectionnant l'action avec la valeur maximale pour chaque état. La table est ensuite triée par ordre croissant des états et un algorithme de recherche par dichotomie a été implémenté en Structured Text (ST). Cette solution permet de limiter la complexité algorithmique nécessaire à la recherche de l'état. Le filtre logique est maintenu à la fin du programme ST pour garantir la sécurité de l'API et contrer les attaques cyber.

3 Preuve de concept

Nous avons réalisé une preuve de concept (PoC) de l'apprentissage à l'implémentation dans un API. Le système étudié se compose de 3 convoyeurs et d'un robot. La commande repose sur la synchronisation de 13 tâches opératives et d'une tâche d'attente volontaire. Pour l'apprentissage nous avons utilisé l'algorithme du Double Q-Learning. Le simulateur de PO utilisé pour cette PoC est Factory IO. La Figure (1) schématise les éléments clés de la méthodologie appliquée sur la PoC. La génération a été réalisée avec 25 épisodes de 10 pièces évacuées (26 min). Les gains sont définis de manière empirique. La tâche finale a un gain de 81, tandis que les tâches définies comme

nécessairement à valeur ajoutée ont un gain de 1. Les autres tâches ont un gain de -25. La table générée comporte 236 états et a été implémentée dans un API sous forme d'un bloc de données (DB).

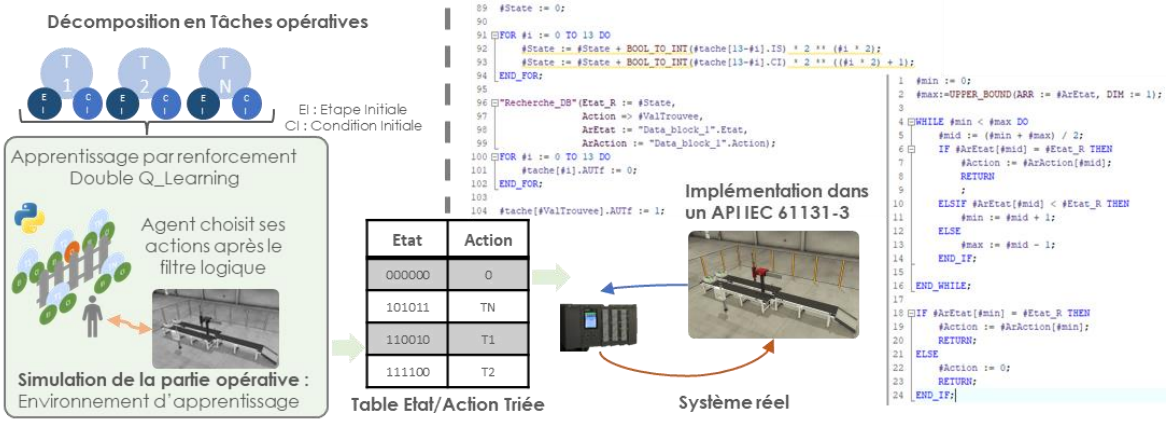

FIG. 1 – Méthodologie d'apprentissage et d'implémentation de la commande

Les algorithmes d'apprentissage par renforcement intègrent une part d'aléatoire afin d'explorer de nouvelles possibilités. Avec les mêmes hyperparamètres, 2 apprentissages peuvent avoir une vitesse très différente pour converger vers la politique optimale. Il est donc difficile de déterminer les hyperparamètres optimaux pour générer une commande.

4 Conclusions

Ce résumé présente une nouvelle méthodologie pour générer et implémenter un modèle d'apprentissage par renforcement de commande dans un API. Les scénarii pour l'apprentissage sont réalisés via un simulateur de PO avec comme état les conditions initiales et l'état initial de chaque tâche opérative du système. L'explosion combinatoire est gérée également par le filtre logique ainsi que l'exécution d'une unique tâche par cycle API. La table, l'algorithme de recherche et les programmes des différentes tâches ont été implémentés dans un API en ST pour contrôler le système.

Références

[1] F. Jaensch, A. Csiszar, A. Kienzlen, et A. Verl, « Reinforcement Learning of Material Flow Control Logic Using Hardware-in-the-Loop Simulation », in *2018 First International Conference on Artificial Intelligence for Industries (AI4I)*, Laguna Hills, USA, 2018, p. 77-80.

[2] F. Jaensch, A. Steidle, et A. Verl, « Curriculum Multi-Stage Reinforcement Learning for Automated Interlinked Production Systems on Virtual Commissioning Simulations », in *2021 Third International Conference on Transdisciplinary AI*, Laguna Hills, USA, 2021, p. 29-136.

[3] M. Kranz, V. Nitsch, et S. Mütze-Niewöhner, « Linking Discrete-Event Simulation with Artificial Intelligence: A Literature-Based Analysis of Existing Approaches in the Context of Manufacturing Planning and Control », in *2023 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Singapore, 2023, p. 1194-1198.

[4] R. Pichard, « Contribution à la Commande des Systèmes à Événements Discrets par Filtre Logique », Thèse de doctorat, Université de Reims Champagne-Ardenne, Reims, 2018.

[5] R. Sutton et A. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[6] J. Zinn, B. Vogel-Heuser, et P. Ockier, « Deep Q-learning for the Control of PLC-based Automated Production Systems », in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, Hong Kong, 2020, p. 1434-1440.