Apprentissage par renforcement avec un réseau de neurones

Une méthode d'apprentissage par renforcement comme celle définie par l'algorithme 9 est adaptée pour traiter des problèmes de taille raisonnable, cependant, quand la taille du problème considéré augmente, des difficultés d'implémentation surgissent.

Premièrement, la mémoire nécessaire pour stocker les valeurs de la fonction Q pour tous les couples état-action visités explose rapidement et deuxièmement, il est possible que le temps nécessaire pour explorer tous les couples état-action afin de caractériser correctement l'évolution de la fonction Q du système ne soit trop important.

Cette deuxième difficulté est particulièrement problématique dans les problèmes à espace d'état continu où il est alors impossible d'expérimenter un nombre infini de fois tous les couples état-action. Pour rappel, si l'algorithme d'apprentissage n'expérimente pas plusieurs fois un même état, les valeurs apprises de la fonction Q ne seront pas représentatives du système étudié.

Pour ces raisons, l'utilisation d'une fonction d'approximation est régulièrement mise en œuvre dans les problèmes où une implémentation tabulaire classique n'est pas envisageable. Comme il a été montré dans les sections précédentes, une fonction d'approximation telle qu'un réseau de neurones est un outil puissant pour approximer des fonctions non linéaires complexes et stocker un grand nombre d'informations dans les coefficients synaptiques.

4.5.1 Exemple pratique des limites d'une implémentation tabulaire

Dans les travaux de [130], les auteurs ont pour ambition d'appliquer un processus d'apprentissage par renforcement à un robot qui recherche une source de lumière : dans un environnement 2-D possédant une source de lumière, un robot-agent tente de développer une stratégie efficace pour trouver l'emplacement de cette source de lumière.

L'agent reçoit ainsi des récompenses positives croissantes lorsqu'il se rapproche de la source de lumière et des récompenses négatives croissantes lorsqu'il s'en éloigne. Deux discrétisations de l'espace d'état sont effectuées pour simuler un problème de faible dimension et un problème de plus grande dimension.

Dans une première approche, un algorithme de Q-learning est implémenté, où les valeurs de la fonction Q sont stockées dans une table. Puis, le même algorithme est développé avec un stockage des résultats dans un réseau de neurones. Les différentes expériences ont montré la supériorité de l'implémentation tabulaire pour le problème de faible dimension alors que l'implémentation neuronale est plus efficace quand la dimension du problème est plus importante.

Bien que le problème explicité dans [130] soit assez simpliste, ces travaux constituent une bonne illustration des limites de l'implémentation tabulaire lorsque la dimension de l'espace d'état augmente.

Ainsi, dans la suite des travaux la vision tabulaire de l'apprentissage par renforcement est abandonnée au profit d'une approche utilisant une fonction d'approximmation, plus connue sous le nom d'*approche connexionniste*.

4.5.2 Approche connexionniste

Dans le cadre d'un apprentissage par renforcement mettant en œuvre un réseau de neurones, l'objectif est d'utiliser le réseau pour effectuer une approximation de quelques unes des caractéristiques essentielles déduites par AR afin d'appliquer une politique décisionnelle au problème considéré.

Généralement, une approximation de fonction est utilisée pour estimer les valeurs de $Q^{\pi}(s,a)$, de cette manière n'importe quel algorithme d'apprentissage par renforcement peut être mis en œuvre sans nécessiter de grosses modifications structurelles.

Dans ces travaux, nous nous intéressons à l'application d'une fonction d'approximation à un algorithme de TD-learning classique de type SARSA. L'estimateur a alors pour but principal d'apprendre la récompense[10] reçue après avoir effectué l'action a dans l'état s en suivant la politique π. Ainsi, le couple $\{s,a\}$ représente les entrées de l'estimateur, tandis que $Q^{\pi}(s,a)$ est la sortie souhaitée.

La structure de l'estimateur peut être déterminée assez librement. En effet, certains travaux utilisent les résultats de l'estimateur pour déterminer la meilleure action à effectuer [129], [130], tandis que d'autres entraînent un estimateur par action possible afin d'étudier chaque action indépendamment des autres [194], [195].

L'une des difficultés majeures de l'approche connexionniste est de savoir comment effectuer l'entraînement du réseau de neurones pour apprendre la politique souhaitée. Dans l'approche tabulaire, l'apprentissage s'effectue en ré-écrivant la valeur de $Q^{\pi}(s,a)$ dans la table de stockage, cependant la manœuvre est plus délicate dans le cas d'un approximateur neuronal puisqu'il est nécessaire de conserver les propriétés de généralisation et la précision sur l'estimation fournie.

La méthode la plus communément utilisée consiste à n'entraîner l'estimateur que pour les couples état-action visités par l'agent, afin d'éviter l'apprentissage de cas non utiles à la résolution du problème. En outre, la discrétisation de l'espace d'état est également primordiale pour garantir de bonnes propriétés d'approximation et de généralisation.

4.5.3 Discrétisation de l'espace état-action

4.5.3.1 Malédiction de la dimension

Dans l'éventualité d'un espace état-action continu, il peut s'avérer nécessaire d'effectuer une discrétisation pour pouvoir représenter l'espace d'état de manière synthétique. Or, le coût de discrétisation de l'espace d'état croît exponentiellement avec la dimension de cet espace, ce qui a pour effet majeur de mener à une explosion de la complexité de résolution du PDM. Ce phénomène est connu sous le nom de *malédiction de la dimension* (ou *curse of dimensionality* en anglais).

En d'autres termes, un compromis doit être effectué entre précision et rapidité de calcul : plus la description des états du système est précise, plus le temps de calcul

10. Ici, l'utilisation du terme *récompense* est un abus de langage, puisqu'il s'agit plutôt de déterminer la profitabilité d'une action effectuée dans un état. Néanmoins, cette profitabilité est directement déduite des récompenses reçues successivement par l'agent.

nécessaire à la résolution du problème est important.

Le raisonnement inverse amène alors à la constatation suivante : pour un problème de dimension donné, le nombre de variables utilisées pour décrire un état doit être suffisamment faible pour ne pas rendre la résolution trop coûteuse en temps de calcul.

Dans la littérature, ce compromis est trouvé en employant des méthodes de discrétisation plus appropriées qu'une simple maillage uniforme de l'espace d'état. [196] et [197] proposent par exemple de réduire la densité du maillage de l'espace d'état dans les zones où une plus faible précision est requise. [190], [198] et [199] proposent quant à eux d'utiliser un codage en grille, dont le principe est d'appliquer un certain nombre de maillages imbriqués pour subdiviser l'espace d'état de l'environnement.

4.5.3.2 Discrétisation de l'espace d'état

La discrétisation de l'espace d'action n'est pas nécessaire puisque le nombre d'actions que peut effectuer un agent est relativement faible : dans le cas de l'étude des temps d'arrêt en station pour une ligne de métro, la précision requise est de l'ordre de la seconde. L'espace d'action A est donc naturellement discret et le cas d'un espace d'action continu n'est pas considéré.

En revanche la discrétisation de l'espace d'état est une nécessité puisque celui-ci est continu : lors de l'exploitation, les trains peuvent occuper une infinité de positions possibles.

La discrétisation de l'espace d'état concerne principalement deux notions : le pas de discrétisation et l'encodage des variables.

Le choix du pas de discrétisation permet de fixer la granularité de l'étude. Dans un problème à espace d'état continu, un pas de discrétisation faible entraîne une meilleure précision sur les données au détriment de la taille de l'espace d'état.

L'encodage des variables est également une notion importante, puisqu'il permet à l'estimateur de faire la distinction entre les différents couples état-action qui lui sont présentés, tout en lui permettant de développer ses capacités de généralisation.

De fait, l'encodage doit adopter une certaine logique pour que les états proches dans la réalité bénéficient d'un encodage mettant en évidence leurs points communs.

Cependant, comme le rappelle [185], la méthode de représentation des entrées est largement dépendante du problème étudié, ce qui impose d'effectuer divers essais et erreurs pour déterminer l'encodage le plus approprié.

Par conséquent, une discrétisation dérivée du codage en grille est utilisée, en exploitant les délimitations naturelles imposées par la présence de stations à des positions fixes.

Un positionnement relatif des trains a été privilégié par rapport à un positionnement absolu afin de faire chuter considérablement le nombre de variables nécessaires pour décrire l'état des trains sur la ligne.

Avec ce procédé de discrétisation, seules trois variables sont nécessaires pour représenter efficacement le positionnement d'un train sur la ligne.

4.5.4 Algorithme connexionniste d'apprentissage par renforcement

Les traces d'éligibilité sont utilisées pour simuler une mémoire des expériences afin d'identifier quels couples état-action visités précédemment sont responsables de la récompense reçue au pas de temps courant.

Dans une approche connexionniste, les traces ne sont plus définies par couples état-action, mais par poids synaptiques : la trace d'éligibilité d'un coefficient synaptique donne alors une indication sur l'impact de chaque coefficient dans la mise à jour de l'algorithme de rétropropagation.

Plusieurs difficultés d'implémentation surviennent.

D'une part, l'implémentation connexionniste est plus contraignante que l'approche tabulaire puisqu'il est nécessaire de modifier l'algorithme de rétropropagation de l'erreur pour intégrer les traces d'éligibilité : la règle de mise à jour de la trace d'éligibilité à l'itération k est de la forme de (4.43), où $\Delta\omega_Q$ est la matrice des coefficients synaptiques de la fonction d'approximation définissant Q. Il est alors nécessaire de stocker les matrices de coefficients synaptiques aux différentes itérations pour appliquer cette règle de mise à jour, ce qui nécessite d'allouer de la mémoire supplémentaire.

$$e_k = \gamma\lambda e_{k-1} + \Delta\omega_Q \tag{4.43}$$

D'autre part, cette approche n'est pas compatible avec un apprentissage supervisé puisque l'algorithme d'apprentissage supervisé n'utilise pas la récompense courante pour calculer les gradients de l'erreur, mais les récompenses présentes dans la base de donnée. La récompense courante n'est donc propagée aux couples état-action visités précédemment que s'ils se trouvent dans la base d'apprentissage courante [185].

Dans la littérature, il existe quelques exemples de travaux ayant utilisés le principe des traces d'éligibilité dans une approche connexionniste incrémentale comme [191] ou [194]. Pour rappel, l'apprentissage online consiste à présenter chaque cas d'apprentissage un par un, ce qui dans le cas d'une base de données déjà constituée demande plus de temps d'apprentissage qu'une approche offline.

Pour ces raisons, nous avons choisi de poursuivre l'étude dans le cas d'un apprentissage offline.

4.5.4.1 Neural fitted Q-iteration

La méthode *Neural Fitted Q-iteration* (NFQ) a été introduite par [200] pour offrir une alternative à la mise à jour en ligne des coefficients synaptiques d'un réseau neuronal. L'apprentissage est ainsi réalisé en offline en collectant l'ensemble des transitions $\{s, a, s', r\}$ issues de simulations d'interactions entre l'agent et l'environnement.

Cela revient à effectuer un apprentissage par renforcement offline puisque toutes les interactions sont connues au préalable : la référence d'apprentissage de la fonction Q pour chaque cas est calculée de manière déterministe pour être ensuite apprise par le réseau neuronal via un algorithme classique d'apprentissage supervisé.

La méthode NFQ présente l'avantage de mener à une phase d'apprentissage plus courte de par l'intégration d'un apprentissage supervisé. Une architecture d'apprentis-

sage similaire à celle décrite par la figure 4.8 peut alors être utilisée pour mettre en œuvre cette méthode.

Cette méthode constitue une première solution pour effectuer un apprentissage par renforcement, cependant, le désavantage de cette méthode est de ne pas garder une mémoire des états visités. Ainsi, au lieu d'appliquer le principe des traces d'éligibilité, il a été choisi d'explorer une autre solution : la méthode *Dyna*.

4.5.4.2 Architecture Dyna

L'architecture Dyna a été proposée par Sutton dans [201] et mise en œuvre notamment dans une implémentation tabulaire dans [202] et [203] afin d'effectuer un compromis entre les méthodes model-based et model-free. Cette architecture est également appelée *Dyna-Q learning*.

Sutton la définit comme une "*architecture intégrée permettant un apprentissage, une planification et une réaction*" [204].

Comme le souligne [205], l'architecture Dyna utilise un modèle partiel et déterministe du système qui est enrichi par de nouvelles expériences à chaque nouvelle itération de l'algorithme.

Dans [201], Sutton suggère d'apprendre un modèle de l'environnement pour déterminer les valeurs de la fonction de récompense et les états consécutifs à la prise d'action dans un état donné.

Grâce à cette approche, l'agent est en mesure d'une part de connaître la conséquence des actions effectuées, mais également d'effectuer une estimation sur les signaux de renforcement pour les actions non explorées.

D'un point de vue pratique, l'architecture Dyna et les traces d'éligibilité ont le même impact sur l'apprentissage en permettant de mettre à jour plusieurs couples état-action à chaque itération.

Les traces d'éligibilité permettent de garder une mémoire temporaire de la trajectoire visitée alors que l'architecture Dyna permet de développer un modèle de l'environnement, ce qui en définitive crée une mémoire à long terme de toutes les expériences et de leur impact respectif sur l'objectif de l'agent.

Dans [206], les auteurs effectuent une comparaison entre une méthode SARSA(λ) et une méthode Dyna-SARSA dans une implémentation tabulaire d'un problème de labyrinthe. Ils montrent que l'architecture Dyna est plus efficace à trouver la politique optimale du problème dès lors que la taille de l'espace d'état augmente.

Par contre l'architecture Dyna présente de moins bonnes performances qu'une méthode intégrant des traces d'éligibilité dans le cas où les états du système sont partiellement observables.

Un algorithme simplifié de la méthode Dyna-Q est proposé par l'algorithme 10, où χ est le modèle de l'environnement, Φ est la fonction dont dérive la politique π et Ω est le modèle de récompense[11].

Il est à noter que certaines versions de Dyna-Q utilisent un même approximateur pour stocker les couples (s', r), cependant l'utilisation d'une fonction d'approximation

11. les variables P, R et Q introduites précédemment sont respectivement remplacées par χ, Ω et Φ afin de marquer l'originalité de la démarche Dyna-Q connexionniste par rapport à la définition classique d'un PDM.

par variable permet une plus grande souplesse et facilite la phase d'apprentissage dans une approche connexionniste.

La première partie de l'algorithme 10 correspond à une routine d'apprentissage de fonction Q d'une méthode NFQ classique avec une phase d'apprentissage des fonctions χ, Ω et Φ, tandis que la seconde partie permet d'intégrer la notion de planification à la fonction Q en exploitant χ et Ω pour mettre à jour la fonction Φ pour des couples état-action déjà visités.

Le modèle de l'environnement est alors substitué à l'environnement réel pour accélérer le processus d'apprentissage [207].

Notons que si m a une valeur nulle, l'algorithme 10 se résume à un apprentissage par renforcement de type Q-learning. L'étape de planification est donc une étape optionnelle pouvant être activée ou désactivée au gré du processus d'apprentissage.

Algorithme 10 Algorithme type Dyna-Q connexionniste

Initialiser la politique $\pi(\Phi)$ suivie par l'agent
Initialiser les fonctions χ et Ω
Pour tout $a \in A$, $s \in S$ **Faire**
    Initialiser la fonction $Q(s, a)$
Fin du Pour
Pour tout épisodes **Faire**
    Initialiser l'état de départ s.
    Tant que L'état final n'est pas atteint **Faire**
        $a \leftarrow \pi(\Phi)$
        Recevoir la récompense r et observer l'état suivant s'
        $a' \leftarrow \pi(\Phi)$
        Mettre à jour la fonction Φ avec la règle (4.37)
        $\chi(s, a) \leftarrow s'$
        $\Omega(s, a) \leftarrow r$
        $s \leftarrow s'$
        Pour m itérations **Faire**
            $s \leftarrow$ Choisir aléatoirement un état déjà visité s
            $a \leftarrow$ Choisir aléatoirement une action a déjà effectuée en s
            $s' \leftarrow \chi(s, a)$
            $r \leftarrow \Omega(s, a)$
            Mettre à jour la fonction Φ avec la règle (4.37)
        Fin du Pour
    Fin du Tant que
Fin du Pour

Dans une approche connexionniste et en supposant un apprentissage réalisé correctement, il peut être supposé que les fonctions χ, Ω et Φ développent des propriétés de généralisation qui permettent d'aller plus loin dans le processus de planification, en effectuant de l'anticipation, notamment en traitant des couples état-action non visités.

Cette propriété n'est exploitée que très rarement dans la littérature comme dans [208], dans le but de développer les capacités d'anticipation de robots.

Trois raisons principales pourraient expliquer cet état de fait : premièrement, il est nécessaire d'utiliser une approche connexionniste pour réaliser l'apprentissage par ren-

forcement du modèle de l'environnement, deuxièmement, l'apprentissage des fonctions χ, Ω et Φ doit permettre d'atteindre un taux d'erreur d'estimation suffisamment faible pour avoir confiance dans la prédiction réalisée sur de nouveaux événements et troisièmement, le temps de calcul nécessaire pour atteindre ce niveau d'erreur peut s'avérer prohibitif dans le cas de problèmes complexes.

En effet, il peut apparaître difficile d'atteindre un taux d'erreur suffisamment faible dans des problèmes complexes de grande dimension, puisque cela supposerait d'avoir visité suffisamment de couples état-action uniformément répartis dans l'espace un grand nombre de fois.

4.5.4.3 Pourquoi utiliser une architecture Dyna neuronale ?

La réalisation des objectifs initiaux de l'optimisation énergétique en temps réel, nous a amené à considérer les trois caractéristiques suivantes.

Capacité d'approximation : l'utilisation d'un réseau neuronal permet de développer des capacités d'approximation souhaitable dans le cas de problèmes présentant un espace état-action dont l'exploration exhaustive est impossible.

De plus le temps de réponse d'un réseau neuronal est négligeable ce qui en fait un bon candidat pour des applications temps-réel.

Rapidité et convergence de l'apprentissage : l'apprentissage supervisé a démontré des capacités particulièrement utiles pour traiter des problèmes complexes en assurant la convergence de l'apprentissage en un temps de calcul raisonnable.

Modèle de l'environnement et de la fonction récompense : la constitution d'un modèle de l'environnement et d'un modèle de la fonction récompense permet de développer des capacités de planification utiles si l'on souhaite limiter la taille de la base de donnée après avoir effectué l'apprentissage d'une politique optimale.

Une fois ces modèles appris, il peut être envisagé de ne mettre à jour la base de donnée d'apprentissage qu'avec de nouveaux cas d'étude simulés ou mesurés afin d'éviter tout sur-apprentissage.

Une méthode Dyna-Q connexionniste intégre ces trois caractéristiques dans son fonctionnement sous l'hypothèse de réaliser un apprentissage supervisé des fonctions d'approximation, ce qui en fait une technique adéquate pour répondre aux enjeux de l'optimisation temps réel.

L'algorithme d'optimisation hybride développé au chapitre précédent est ainsi mis à profit afin d'ajouter la notion d'apprentissage supervisé à la méthode Dyna-Q pour créer une base de données permettant d'orienter l'apprentissage plus rapidement vers une politique conforme aux objectifs visés.

Cette méthode nommée *Dyna-NFQ* (DNFQ) s'inspire du processus décrit par l'algorithme 10 et y rajoute une base d'apprentissage regroupant un ensemble d'épisodes dont les caractéristiques sont connues.

4.5.5 Méthode Dyna-NFQ

4.5.5.1 Batch training

Contrairement à l'apprentissage réalisé dans le cadre de l'estimateur neuronal des flux de puissances, l'apprentissage par renforcement nécessite une mise en œuvre un peu différente bien que dans les deux cas l'apprentissage soit supervisé, notamment parce qu'ici il est nécessaire d'aller plus loin que le simple apprentissage d'une base de données, il est également nécessaire de développer une politique efficace utilisable pour des cas non présents dans la base de données.

Un apprentissage online peut être envisagé, mais présente le désavantage majeur que lors des premières itérations, la politique construite ne soit pas représentative des données futures, ce qui rend le processus d'apprentissage progressif et donc plus long.
A l'inverse, en connaissant par avance l'ensemble des cas de la base d'apprentissage, il est possible dès les premières itérations de l'apprentissage de construire une politique tenant compte de toutes les caractéristiques connues du système étudié.

En reprenant l'analogie avec un jeu de plateau, un apprentissage offline est assimilable à un apprentissage où l'on connaît d'avance toutes les règles du jeu, tandis que dans un apprentissage online, l'agent apprenant ne reçoit les informations concernant les règles du jeu que progressivement au cours des parties jouées, ce qui bien évidemment rend les politiques apprises lors des premières parties inefficaces.

Malgré tout, la progressivité de l'apprentissage online est une caractéristique intéressante pour effectuer un apprentissage sur une base de données de grande taille. Dans la littérature l'adaptation de cette caractéristique à un apprentissage est appelée *batch training*.
Les données d'apprentissage sont donc partitionnées en sous-groupes (batch) de manière à faciliter l'apprentissage. Chaque sous-groupe de données est présenté itérativement au réseau neuronal jusqu'à ce que l'erreur d'estimation atteigne un niveau acceptable.
Enfin, le batch training permet au réseau neuronal de passer régulièrement en revue des cas appris précédemment pour ne pas que l'agent apprenant oublie ce qu'il a déjà appris.

4.5.5.2 Hint to the goal

Dans le cadre d'un apprentissage par renforcement, l'apprentissage des cas défavorables est presque aussi important que celui des cas favorables, puisqu'il est souhaitable que durant la phase d'exploitation l'agent puisse choisir des actions permettant de maximiser les récompenses reçues, mais également que durant la phase d'exploration, il puisse faire l'expérience de "mauvaises" situations soit pour les éviter par la suite, soit pour les utiliser afin de se retrouver dans un état plus favorable dans les transitions suivantes.

En outre, dans le cas de l'apprentissage supervisé classique d'un réseau neuronal, Riedmiller évoque l'utilisation de cas artificiels d'apprentissage pour forcer le réseau neuronal à reconnaître les états terminaux (ou états favorables) en leur assignant une valeur de coût en conséquence [200], [209].

L'auteur donne à cette méthode le nom de *hint to the goal*, puisqu'elle permet d'indiquer une zone de l'espace comme la région à atteindre pour réaliser l'objectif de l'apprentissage.

Le principe inverse peut donc être en théorie applicable pour forcer le réseau neuronal à reconnaître les états défavorables ou ne menant pas aux récompenses maximales.

Cela consiste à générer des cas d'apprentissage sur la base d'interactions simulées d'un agent avec son environnement et à ne pas écarter les solutions jugées comme mauvaises afin de s'assurer que la base d'apprentissage contienne à la fois des indications sur les *bonnes* et *mauvaises* situations.

En pratique, ces mauvaises solutions sont obtenues lors des premières itérations de l'optimisation hybride OEP-AG (cf. figure 3.4.8).

4.5.5.3 Observations empiriques

En outre, la réalisation de multiples simulations d'apprentissage a permis de mettre en lumière certaines contraintes empiriques :

— L'augmentation de la taille du cache doit être progressive au cours des itérations. Un cache est défini comme la liste de données qui est réellement présentée à l'agent apprenant.

  Ainsi, en fin de cycle d'apprentissage, la taille du cache est égale à celle du batch. L'utilisation d'un cache dont la taille est incrémentée au fil des itérations permet d'assurer la progressivité de l'apprentissage afin d'éviter les modifications trop importantes de la politique apprise.

— Une règle de remplacement des données d'apprentissage doit être appliquée lorsque la taille maximale de la base de donnée est atteinte.

  Malgré les progrès technologiques de ces dernières années sur les coûts du stockage de données et de la puissance de calcul, il est toujours nécessaire de limiter la taille de la base d'apprentissage pour s'assurer que l'apprentissage soit réalisé en un temps raisonnable.

  La forme de la règle de remplacement (4.44) s'inspire de celle développée par [210].

  Cette règle de remplacement insère un nouveau cas dans la base d'apprentissage en remplaçant le cas le plus similaire à ce nouveau cas. ξ_1 et ξ_2 sont respectivement les coefficients de pondération permettant de régler l'importance de la distance entre les états s et les actions a.

$$\vartheta = \sqrt{\xi_1(a_{new} - a_{old})^2 + \xi_2(s_{new} - s_{old})^2} \tag{4.44}$$

— Il a été constaté qu'en ordonnant les données contenues dans le cache avant de les faire apprendre au RNA l'erreur d'apprentissage décroissait plus rapidement qu'en effectuant un batch learning aléatoire.

— Un réapprentissage ponctuel des cas menant aux états les plus favorables permet également de guider la prise de décisions au fil des itérations vers une politique maximisant les récompenses.

4.5.5.4 Implémentation de la méthode Dyna-NFQ

Les macro-étapes de la mise en œuvre de la méthode Dyna-NFQ sont décrites par la figure 4.23. La première étape consiste à créer une base de données contenant les caractéristiques $\{s, a, r, s'\}$ d'un grand nombre d'épisodes.

Ensuite, les couples $\{s, a\}$ sont utilisés pour entraîner les fonctions d'approximation χ, Ω et Φ. Les références s'_{ref} et r_{ref} sont directement issues de la base de données, tandis que Q_{ref} est approché par un critère de récompense actualisé similaire à (4.27).

L'apprentissage supervisé des fonctions d'approximation χ, Ω et Φ est réalisé en appliquant les préconisations de partitionnement de la base d'apprentissage afin de réduire au maximum le temps de convergence des fonctions vers une erreur d'estimation faible et la définition d'une politique optimale.

En outre, la méthode DNFQ étant basée essentiellement sur un apprentissage supervisé d'une base de données, la majorité des points explicités dans la partie concernant la conception d'un estimateur neuronal peuvent être repris comme la normalisation des données d'apprentissage, le suivi de l'erreur d'apprentissage ou encore l'étude des performances des fonctions d'approximation.

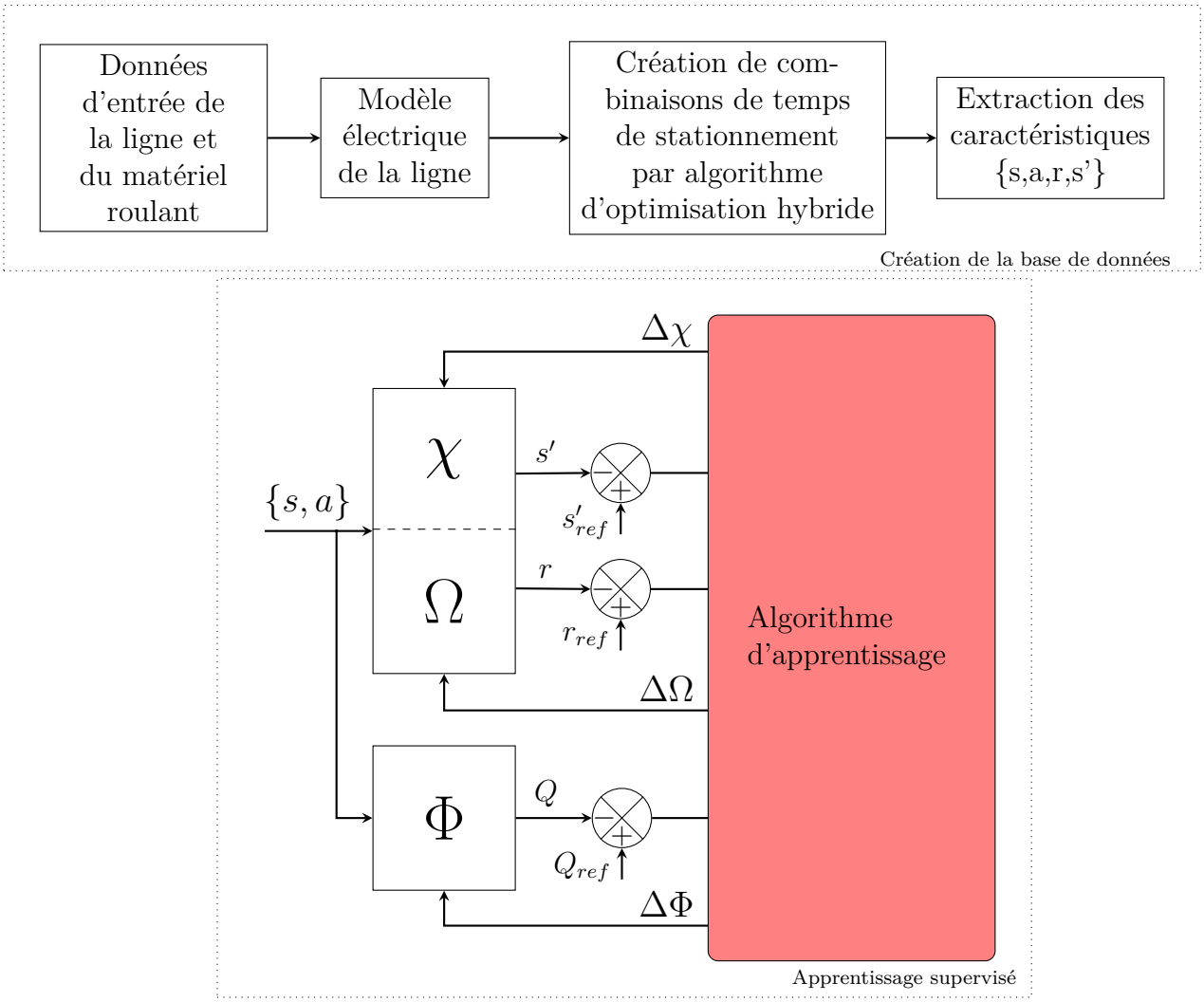

FIGURE 4.23 – Structure de la méthode Dyna-NFQ.

4.5.6 Robustesse de la méthode face aux perturbations

4.5.6.1 Étude des aléas de trafic

La base d'apprentissage générée par l'algorithme d'optimisation hybride joue un rôle prépondérant dans le fonctionnement de la méthode DNFQ. Elle doit être suffisamment représentative du comportement moyen de la ligne de métro pour s'assurer que la fonction valeur et le modèle de l'environnement appris restent vrais dans le cas de perturbations.

Par extension, il peut être intuitivement spécifié que plus le nombre de cas de simulations est important plus il y a de chance de rencontrer des cas d'exploitation qui pourraient survenir en exploitation réelle, sous l'hypothèse de laisser aux algorithmes d'optimisation une marge de manœuvre suffisamment importante pour simuler des cas d'exploitation intégrant des aléas.

Les différents enregistrements réalisés lors d'essais-site sur la ligne de Turin ont permis de mener une étude statistique de la nature des aléas d'exploitation les plus probables et de leur fréquence moyenne d'occurrence. Ces résultats sont synthétisés par les figures 4.24, 4.25 et 4.26.

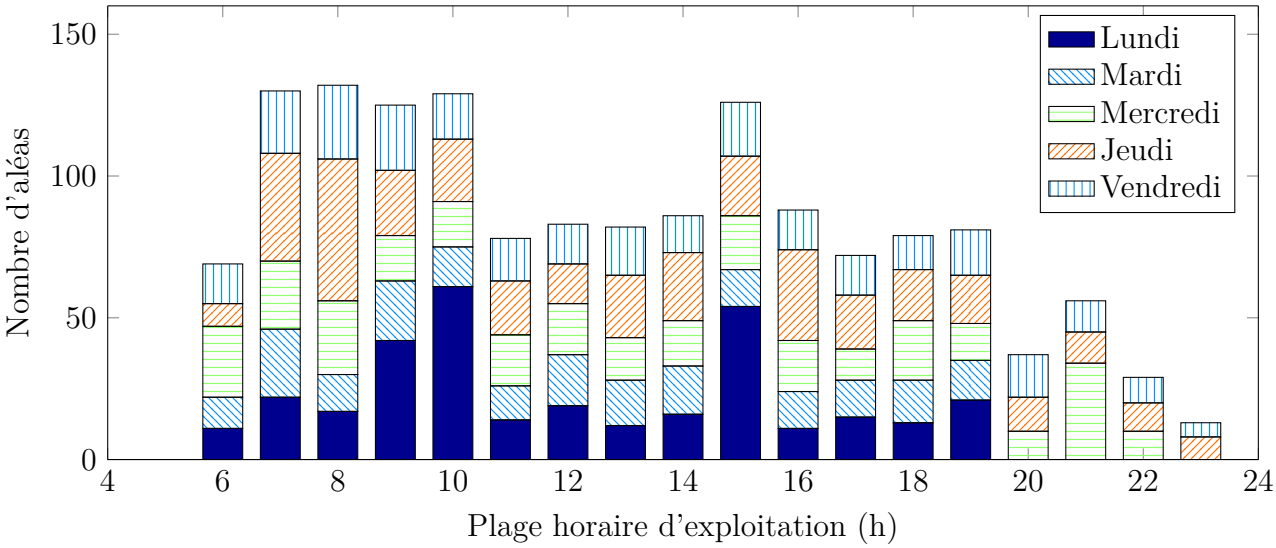

FIGURE 4.24 – Distribution des aléas hebdomadaires en fonction de la plage horaire d'exploitation

D'après la figure 4.26, environ 75% des aléas hebdomadaires ont une durée inférieure ou égale à 2s.

Ainsi, le paramétrage de l'algorithme d'optimisation hybride doit être défini de manière à explorer des solutions représentatives de la distribution des aléas afin de pouvoir recréer une dispersion des temps de stationnement analogue. De cette manière, la méthode DNFQ est en mesure d'intégrer ces conditions de fonctionnement dans les modèles appris et de développer une politique robuste permettant de re-synchroniser les trains dans n'importe quelle configuration de carrousel.

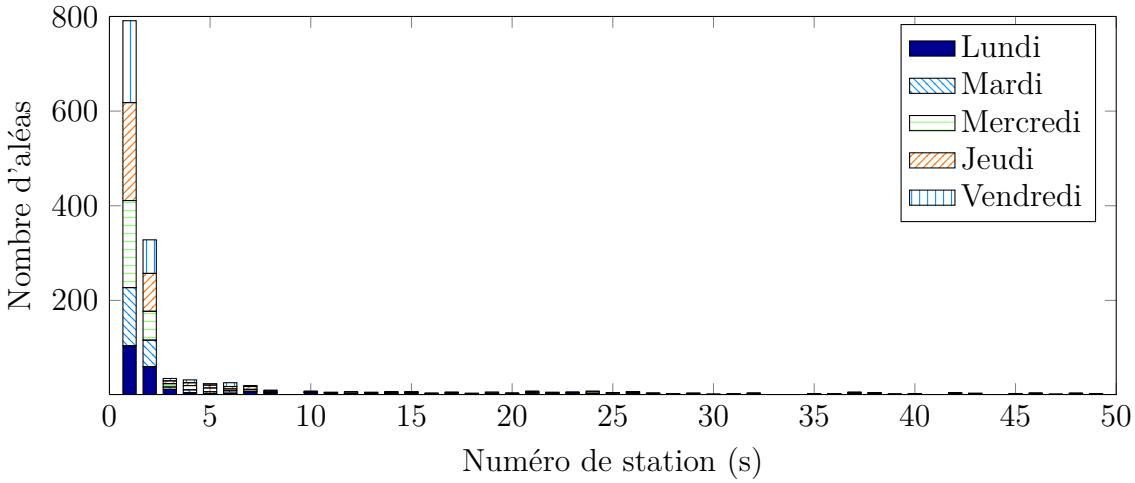

FIGURE 4.25 – Distribution de la durée des aléas d'exploitation hebdomadaires

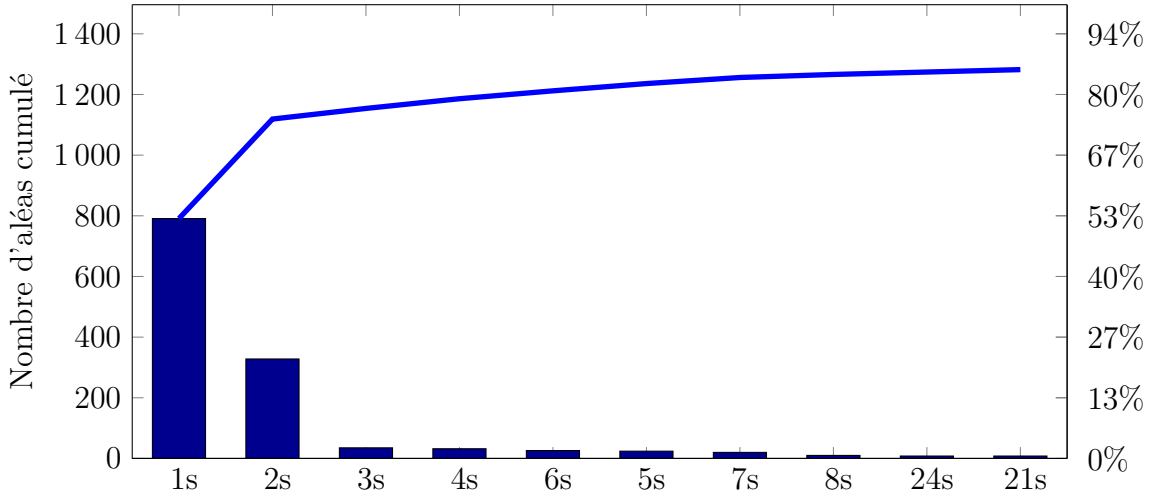

FIGURE 4.26 – Distribution de pareto de la durée des aléas hebdomadaires

4.5.6.2 Étude de robustesse

Dans [211], les réseaux de neurones sont définis comme tolérants aux fautes ou robustes de par leurs propriétés intrinsèques. Dans notre cas d'étude, la propriété la plus intéressante est *la nature distribuée de l'information* : du fait du grand nombre de neurones dans un réseau, chaque neurone contribue dans une faible proportion à la réponse finale. De cette propriété découle la capacité de généralisation des RNA [212].

Dans [211], il est fait mention de deux types de fautes : une dégradation interne des informations contenues dans le RNA (une perte d'information sur les poids synaptiques ou la suppression d'un neurone) et le bruit des données d'entrées. C'est ce dernier type de fautes qui est le plus susceptible de se produire puisque les données d'entrées présentées au RNA sont dépendantes des aléas de trafic.

En outre, comme le montre [213], la robustesse d'un réseau de neurones est améliorée en lui présentant des cas d'apprentissage entachés de bruit.

La robustesse de la politique est alors obtenue en considérant suffisamment de cas de fonctionnement dégradé de la ligne de métro pour que ces points de fonctionnement

soient considérés comme des cas de fonctionnement nominaux.

Pour éprouver la robustesse de la méthode deux cas d'études sont considérés : un cas d'exploitation en heure creuse et un cas d'exploitation en heure de pointe.

Les fréquences ϑ_f et amplitudes ϑ_a des aléas dans chacun des deux cas sont définies selon les conditions d'exploitation moyennes enregistrées lors des essais sur la ligne de Turin aux périodes correspondantes.

La figure 4.27 présente la distribution de l'amplitude ϑ_a des aléas dans chacun des deux cas. Les fréquences d'occurrence des aléas sont respectivement $\vartheta_{f_1} = 260s$ $\vartheta_{f_2} = 145s$.

Ces deux cas de figures sont étudiés pour un même carrousel avec des conditions initiales identiques afin d'observer les performances de la méthode DNFQ lorsque la fréquentation de la ligne varie.

En heure de pointe, du fait d'une fréquentation de la ligne plus importante, les aléas sont plus récurrents qu'en heure creuse, ce qui correspond à un cas défavorable pour la méthode DNFQ.

En outre, l'écart-type de l'amplitude des aléas en heure de pointe est volontairement augmenté pour simuler des perturbations telles que l'algorithme doit suivre la consigne de temps de stationnement minimal pour résorber le retard des trains concernés par les aléas. Ainsi, le retour à un état optimisé doit être coordonné en modifiant les temps de stationnement de plusieurs trains arrivés en station consécutivement.

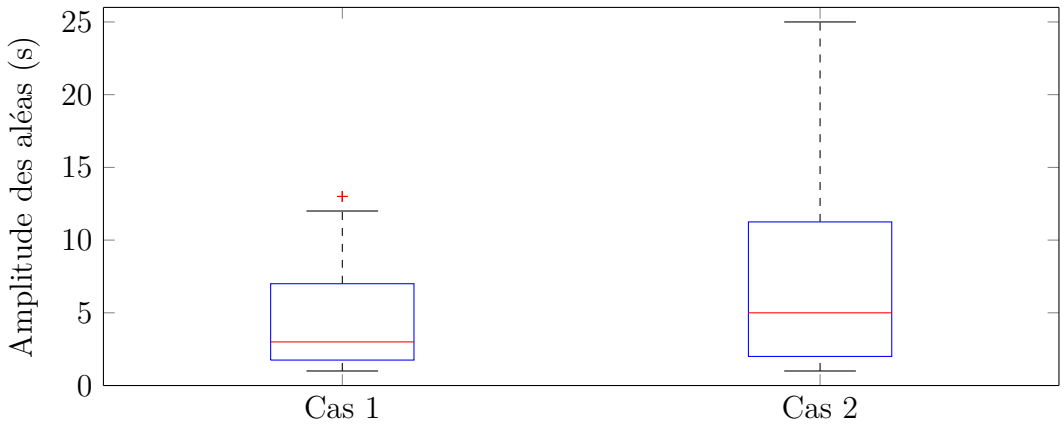

FIGURE 4.27 – Distribution des amplitudes des aléas dans les deux cas d'études.

Cette comparaison permet ainsi d'évaluer la capacité de la méthode à pouvoir redéfinir la politique décisionnelle grâce à la phase de planification : les modèles χ et Ω du système sont utilisés pour créer de nouveaux cas d'apprentissage tenant compte de la nouvelle distribution des aléas afin de modifier la politique Φ.

4.5.6.3 Performances de la méthode DNFQ

La base de données d'apprentissage est constituée en appliquant la méthode d'optimisation hybride OEP-AG. La distribution des solutions obtenues est illustrée par la figure 4.28.

Une très forte densité de solutions générant un gain énergétique supérieur à 10% a été produite en laissant l'algorithme d'optimisation évoluer sur un grand nombre

d'itérations, cela permet ainsi d'orienter l'apprentissage d'une politique menant à des décisions favorables pour la réutilisation de l'énergie issue du freinage récupératif.

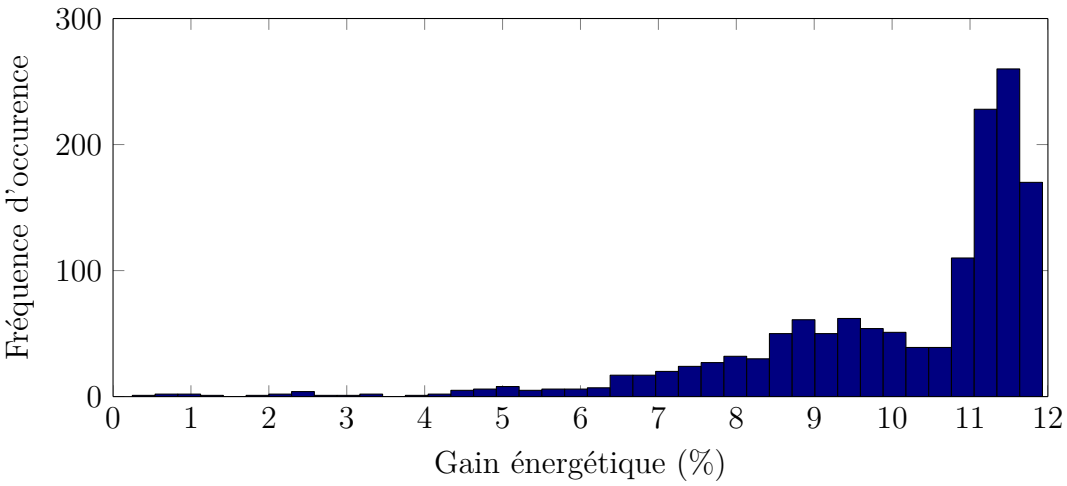

FIGURE 4.28 – Distribution des solutions constituant la base d'apprentissage.

La figure 4.29 présente l'évolution de la consommation énergétique du carrousel sur un tour de boucle dans chacun des deux cas d'études. A chaque itération, la politique courante est appliquée pour un carrousel effectuant un tour de boucle complet. De plus, un même profil d'occurrence des aléas est employé à chaque itération.

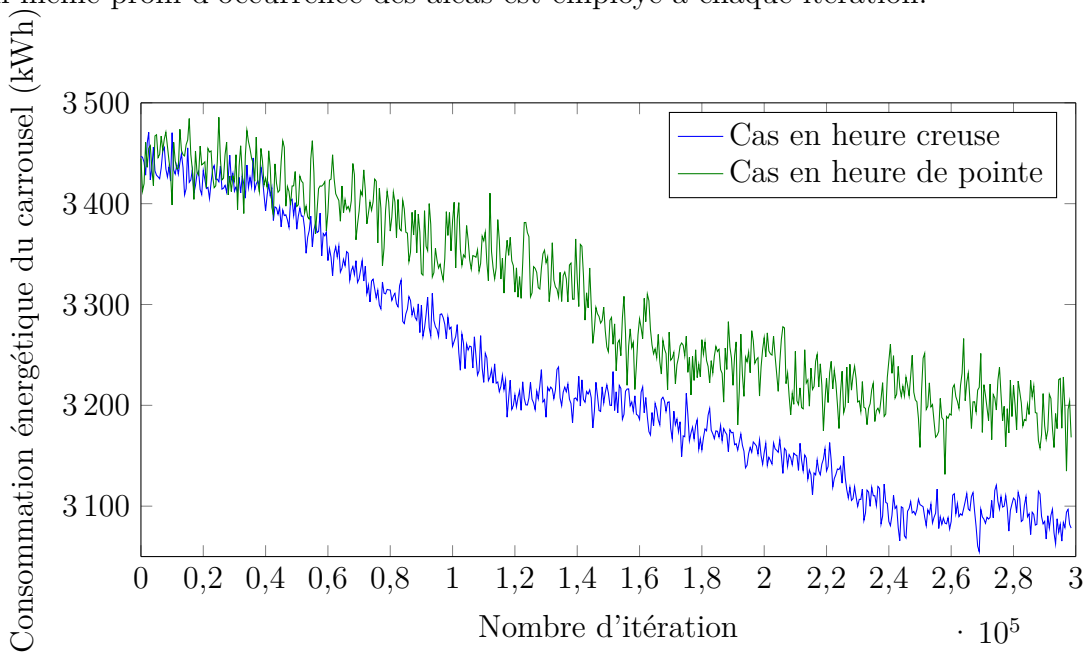

FIGURE 4.29 – Évolution de la consommation énergétique d'un carrousel induite par la politique de stationnement suivie dans deux cas d'études.

Les performances de la méthode DNFQ dans chacun des deux cas d'études de perturbations sont synthétisées dans le tableau 4.4.

Le cas de référence correspond à une exploitation où les trains circulent sans aléas et en suivant la table horaire nominale. Il ne s'agit donc pas d'un cas d'exploitation optimisée, mais de la consommation de base d'un carrousel suivant la consigne nominale initiale fournie par l'exploitant.

Concrètement, il s'agit de la consommation énergétique d'un carrousel sur un tour de ligne en suivant une table horaire nominale et en considérant un cas d'exploitation idéal.

Les autres cas d'études prennent ce cas comme référence dans le calcul des gains énergétiques.

	cas de référence	**cas 1**	**cas 2**
Gain énergétique final	0%	10,3%	7.5%
Temps moyen de prise de décision	✗	$0,468ms$	$0,512ms$
Temps total d'apprentissage	✗	94h15	95h02

TABLEAU 4.4 – Comparaison des performances de la méthode DNFQ dans 2 cas d'exploitation avec perturbations.

Concernant les cas d'étude liés à l'utilisation de la méthode DNFQ (cas 1 et cas 2), l'apprentissage a été limité à un nombre fini d'itérations afin de garder un temps d'apprentissage inférieur à 96h, mais également pour être en mesure de comparer l'évolution de la politique dans les deux cas de figures.

Les temps de calcul correspondent à un PC embarquant un processeur Intel Xeon W3520 cadencé à 2.67GHz, 12 Go de RAM, une carte graphique Quadro FX 3800 avec 1Go de VRAM et le tout sous un environnement Windows 7. Notons que le temps de prise de décision dans le cas du cas de référence n'est pas spécifié, puisque dans ce cas, la décision est connue en amont de la simulation.

Dans le cas où l'exploitation est perturbée par de faibles aléas, le gain énergétique généré par la méthode DNFQ est de 10,3%, tandis que lorsque les perturbations de trafic sont 80% plus fréquentes et de plus grande amplitude, le gain énergétique généré est de 7,5%.

D'après l'évolution des courbes de la figure 4.29, il semble que les performances des politiques décisionnelles issues de la méthode DNFQ pourraient encore être améliorées en permettant une phase d'apprentissage encore plus conséquente.

4.5.6.4 Comparaison par rapport à l'optimisation hors-ligne

Les résultats présentés dans le tableau 4.4 sont à contraster par rapport aux performances d'une table horaire figée dans le cas d'une exploitation soumise à des aléas.

Deux nouveaux cas d'études sont donc définis pour étudier l'impact de l'utilisation de table horaire fixe optimisée pour gérer des aléas en suivant les distributions de perturbations analogues aux cas 1 et 2.

L'inconvénient majeur de cette approche est que selon les conditions de perturbations, une table horaire peut naturellement permettre de résorber les aléas ou favoriser les phases de synchronisation entre accélération et freinage des trains.

Ainsi, en préambule de cette étude, les 50 meilleures tables horaires optimisées issues de la base d'apprentissage (figure 4.28) sont sélectionnées. Puis les gains énergétiques issus de l'application de ces tables horaires dans les deux conditions de perturbations sont analysés.

Ces résultats sont ensuite moyennés afin d'évaluer le comportement énergétique moyen de tables horaires optimisées en hors-ligne dans un environnement dynamique

soumis à perturbations [12].

Les résultats de cette étude sont repris dans le tableau 4.5, les cas 3 et 4 correspondent respectivement aux conditions de perturbations des cas d'étude 1 et 2.

	cas de référence	**cas 3**	**cas 4**
Gain énergétique moyen	0%	3,2%	0,9%
Gain énergétique maximal	0%	6,4%	2,2%
Gain énergétique minimal	0%	-1,3%	-4,1%

TABLEAU 4.5 – Comparaison des performances de l'optimisation hors-ligne dans 2 cas d'exploitation avec perturbations.

Dans le tableau 4.5, l'explicitation des gains moyens et extrêmes permet de montrer que des tables horaires optimisées en hors-ligne (générant des gains énergétiques élevés dans des cas idéaux d'exploitation) peuvent entrainer une augmentation de la consommation énergétique globale de la ligne par rapport à l'utilisation d'une table horaire nominale, lorsque des aléas d'exploitation surviennent.

Ainsi, de cette étude, il est assez compliqué d'estimer l'impact d'une optimisation hors-ligne sur un problème dynamique temps-réel puisque les gains énergétiques perçus dépendent de la table horaire initiale utilisée. En effet, contrairement à une méthode mathématique fournissant un même optimum global à chaque évaluation, la méthode hybride OEP-AG fournit à chaque évaluation un optimum différent.

Cependant, l'analyse des tableaux 4.4 et 4.5 indique d'une part que l'utilisation d'une optimisation hors-ligne pour solutionner un problème temps réel s'avère aléatoire puisque des gains négatifs peuvent être obtenus, et d'autre part, ce type d'approche est moins efficace qu'une approche de type DNFQ, en considérant l'espérance de gain énergétique pour chaque approche.

4.6 Conclusion

Dans le chapitre précédent, une méthode d'optimisation hybride a été définie pour effectuer la modification des temps de stationnement dans un cas idéal d'exploitation. Dans ce chapitre, nous souhaitions donc adapter cette méthode d'optimisation pour effectuer une resynchronisation en temps réel des trains.

Il était ainsi nécessaire de diminuer le temps de calcul de la boucle d'optimisation tout en conservant sa précision.

Un état de l'art des travaux sur la replanification temps réel dans le domaine ferroviaire a mis en évidence qu'il est indispensable de disposer de modèles énergétiques précis de la consommation des trains pour assurer l'optimalité de la prise de décision.

Or, le temps de calcul de la boucle d'optimisation est expliqué à 90% par le calcul des flux de puissances vu par trains à l'aide d'une méthode de résolution itérative pour connaître la consommation énergétique de la ligne.

12. Pour plus de réalisme, lorsque l'avance/retard des trains par rapport à l'horaire initial est trop important, des règles de régulation sont appliquées pour modifier les valeurs nominales de temps de stationnement afin de ne pas violer les contraintes d'exploitation. Ces règles sont définies de manière à garantir un temps de battement minimal.

De fait, la première partie de ce chapitre a été dédiée à la synthèse du processus de résolution itératif par un réseau de neurones artificiels (RNA). Les RNA ont démontré dans un grand nombre d'applications leur capacité à apprendre le fonctionnement d'un système et de fournir une réponse à une sollicitation en un temps réduit.

Après avoir rappelé la théorie mathématique sur les réseaux de neurones et défini les paramètres d'implémentation, les performances de l'estimateur neuronal ont été testées pour des bases de données composées respectivement de 3.10^3 et 4.10^4 cas d'apprentissage.

Le temps de calcul est amélioré d'un facteur 130 dans le premier cas d'étude, contre un facteur 75 dans le deuxième cas de figure. La précision de l'estimateur est quant à elle dépendante du temps alloué à l'apprentissage du RNA, ainsi plus l'apprentissage est long et plus l'estimateur neuronal est capable d'affiner sa précision et d'augmenter ses capacités de généralisation.

Ensuite, il a été décidé d'appliquer le même principe pour effectuer l'apprentissage de la méthode d'optimisation hybride OEP-AG. La méthodologie de réalisation est un peu différente de celle employée pour concevoir un estimateur neuronal, puisque cette fois, nous souhaitons non seulement définir une politique décisionnelle en fonction des cas d'apprentissage issus de l'optimisation hors-ligne, mais également que cette politique puisse s'adapter en cas de perturbations de trafic.

Pour ce faire, nous avons choisi de mettre en œuvre une méthode basée sur l'apprentissage par renforcement, qui consiste à distribuer des récompenses en fonction des décisions prises. Ainsi, la politique décisionnelle optimale est obtenue en effectuant les actions qui maximisent les récompenses reçues sur un horizon temporel.

Une étude de différentes applications utilisant le principe d'apprentissage par renforcement nous a permis de définir la méthode DNFQ, qui est une méthode hybride faisant intervenir l'apprentissage supervisé de plusieurs RNA utilisés conjointement pour générer de nouveaux cas d'apprentissage afin de développer les capacités de planification de la politique décisionnelle.

Les performances de la méthode DNFQ ont ensuite été testées pour deux distributions d'occurrence d'aléas. Ces distributions sont conçues de manière à simuler un cas présentant des aléas de faibles amplitudes et un deuxième cas où les perturbations ont une amplitude et une fréquence beaucoup plus importantes. A l'issue de la phase d'apprentissage, dans le premier cas de figure, la méthode DNFQ génère une politique permettant un gain énergétique de 10,3%, tandis que dans le deuxième cas, le gain constaté est de 7,5%.

En outre, le temps de prise de décision est de l'ordre de 0,5 ms, ce qui rend possible l'implémentation de cette méthode pour une application temps réel.

Ces résultats sont ensuite contrastés par l'étude des performances de tables horaires issues d'une optimisation hors-ligne pour effectuer la gestion énergétique d'une ligne dans des conditions réelles d'exploitation. Il en ressort que ce type d'approche n'est pas déterministe puisque la consommation énergétique globale d'une ligne peut ainsi se retrouver dégradée par rapport à l'utilisation d'une table horaire nominale.