L'apprentissage par renforcement appliqué au routage

4.1 Introduction

Les problèmes des réseaux informatiques sont complexes, que ce soit le routage, la classification du trafic ou la prédiction de pannes. L'apprentissage automatique peut permettre de résoudre ces problèmes. L'apprentissage automatique est composé de plusieurs catégories. Les trois catégories principales sont l'apprentissage supervisé, non supervisé et par renforcement. Dans le cas de l'apprentissage supervisé, l'objectif est de créer un modèle à partir de données labellisées. L'apprentissage supervisé peut être utilisé pour effectuer de la classification de trafic [Pac+19]. Dans le cas de l'apprentissage non supervisé, l'objectif est de créer un modèle à partir de données non labellisées. Il peut être utilisé par exemple pour la détection d'anomalies [Fal+19]. Contrairement aux deux autres approches, le modèle est créé par essais et erreurs (*try and error*) pour l'apprentissage par renforcement. Il ne nécessite donc pas de jeu de données initial d'entraînement, ce qui est un atout. Les problèmes de routage peuvent être résolus grâce à l'apprentissage par renforcement comme nous allons le voir par la suite. Nous allons commencer par présenter plus en détail l'apprentissage par renforcement (Section 4.2). Nous verrons ensuite des travaux utilisant l'apprentissage par renforcement pour résoudre des problèmes des réseaux informatiques. Enfin, nous terminerons par Q-routing [BL94], un algorithme de routage inspiré de l'algorithme d'apprentissage par renforcement Q-learning [WD92] (Section 4.4).

4.2 Apprentissage par renforcement

Nous allons commencer par définir un peu plus en détails l'apprentissage par renforcement. L'apprentissage par renforcement est le troisième paradigme d'apprentissage automatique. Contrairement à l'apprentissage supervisé et non supervisé, il ne nécessite pas nécessairement de jeu de données initial. L'apprentissage s'effectue à partir de ses expériences (*try-and-error*).

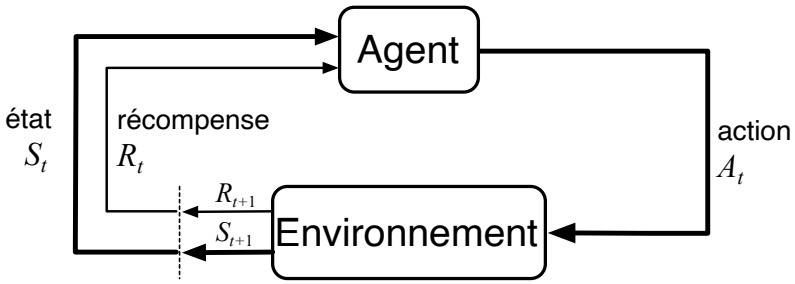


FIG. 4.1 : Interaction entre l'agent et l'environnement dans un processus de décision Markovien, extrait de [SB18].

L'apprentissage par renforcement définit quatre notions fondamentales : l'agent, l'environnement, l'action et la récompense. Les interactions entre ces quatre notions sont illustrées par la Figure 4.1 qui résume le cadre d'un processus de décision Markovien (MDP). De manière pédagogique, nous allons utiliser une analogie d'un joueur dans une salle remplie de machines à sous [SB18]. Ce problème est aussi connu sous le nom de bandit manchot. Le joueur est notre agent. La salle remplie de machines à sous est notre environnement. Les machines à sous composent l'ensemble des états de l'environnement. Une action consiste à jouer sur une des machines à sous. La récompense est le gain obtenu. Chaque machine a une espérance de gain qui lui est propre. L'objectif dans ce cas consiste à maximiser les gains cumulés soit à long terme, soit sur une période (en nombre d'essais).

L'apprentissage par renforcement définit quatre notions supplémentaires : la fonction de valeur, la stratégie, le signal de récompense et le modèle. Nous reprenons notre analogie précédente. Comme l'espérance de chaque machine n'est pas connue, elle peut être au mieux estimée. La fonction de valeur est le nom de la fonction associant une machine à sous et l'estimation de son espérance. Le processus de choix de la machine à sous compose la stratégie de l'agent. Par exemple, la stratégie gloutonne (*greedy*) consisterait à toujours choisir la machine à sous dont l'espérance estimée est la plus élevée. Le signal de récompense est l'objectif de gain à réaliser. Le modèle permet d'effectuer des prédictions sur le comportement de l'environnement, notamment ses réactions par rapport à une action et l'état suivant. Certains algorithmes comme Q-learning ne nécessitent pas de modèle de l'environnement.

Dans une stratégie, il y a deux phases possibles : la phase d'exploration et la phase d'exploitation. Ces deux phases ont une finalité différente. La phase d'exploration sert à améliorer l'estimation des récompenses des différentes actions. L'objectif de la phase d'exploitation est uniquement d'atteindre l'objectif. Nous reprenons notre exemple de bandit manchot. Durant la phase d'exploration, le joueur va essayer une ou plusieurs machines à sous afin d'améliorer l'estimation de l'espérance des gains de celles-ci. Durant la phase d'exploitation, le joueur va utiliser la machine à sous permettant de maximiser les gains. Le passage d'une phase à l'autre est dépendant de la stratégie. Une stratégie gloutonne n'est composée que d'une phase d'exploitation.

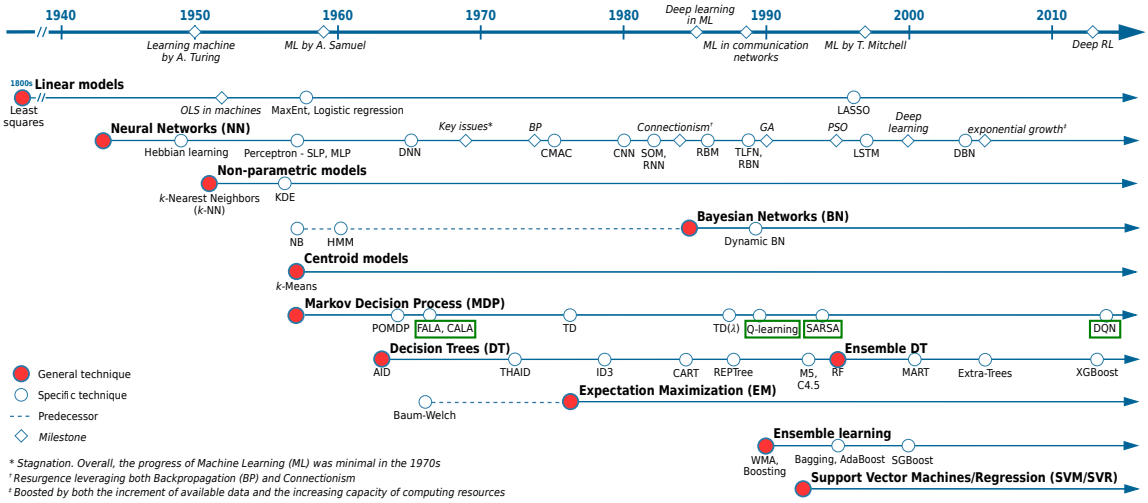


FIG. 4.2 : Développement de l'apprentissage automatique extrait de [Bou+18]. Les algorithmes cités dans ce chapitre sont encadrés en vert.

4.2.1 Quelques exemples d'algorithmes d'apprentissage par renforcement

Nous allons nous intéresser à quelques algorithmes d'apprentissage par renforcement dont Q-learning et ses dérivés. La modélisation d'un réseau informatique est très compliqué, nous avons choisi Q-learning car il ne nécessite pas de modèle, ni de jeu de données initial. La Figure 4.2 illustre le développement de l'apprentissage automatique ci-dessus. Tous les algorithmes présentés ou cités sont encadrés en vert. Ils appartiennent à la catégorie des processus de décision Markovien.

Q-learning

Q-learning est un algorithme d'apprentissage par renforcement créé pendant la thèse de doctorat de Watkins en 1989 et présenté en 1992 [WD92]. Q-learning définit A l'ensemble des actions a, S l'ensemble des états s de l'environnement et R l'ensemble des récompenses r. Afin de contrôler l'apprentissage, Q-learning utilise deux paramètres : le taux d'apprentissage α et le facteur dégressif γ. Le taux d'apprentissage α permet de pondérer la récompense actuelle. Plus la valeur du taux d'apprentissage α est grande, plus Q-learning va donner de l'importance aux dernières récompenses obtenues. Le facteur dégressif γ pondère l'influence de l'état suivant dans le calcul de la récompense de l'état actuel. Plus le facteur dégressif est proche de 1, plus le choix de Q-learning sera influencé par le prochain état. En pratique, le prochain état n'est pas toujours connu. Dans ce cas, le facteur dégressif est nul. Enfin, la Q-fonction est la fonction d'association action-valeur. Il s'agit d'une extension de la fonction de valeur. La Q-fonction associe un état et une action à l'estimation de l'espérance des récompenses. Q-learning définit la Q-fonction telle que :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{4.1}$$

où A_t représente l'action sélectionnée à l'instant t, S_t l'état à l'instant t et R_t la récompense de l'action A_t à l'état S_t.

Algorithme 1 : Algorithme de Q-learning, tel que présenté dans [SB18]

Initialiser $Q(s, a)$ pour tout état $s \in S$, toute action $a \in A$ de façon arbitraire, sauf $Q(\text{état terminal}, \cdot) = 0$;
répéter
  Initialiser S ;
  répéter
    Choisir A depuis S d'après la stratégie et Q ;
    Exécuter A ;
    Observer R et S' ;
    $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \text{max}_a Q(s', a) - Q(S, A)\right]$;
    $S \leftarrow S'$;
  jusqu'à S *soit l'état terminal* ;
jusqu'à *fin* ;

La fonction d'association action-valeur optimale est notée q_*. q_* permettrait de choisir les actions optimales en fonction de chaque état si elle était connue. La Q-fonction approxime directement q_*. Son fonctionnement est décrit par l'Algorithme 1. Le principal avantage de Q-learning est qu'il ne nécessite pas de modèle. Un des inconvénients de Q-learning réside dans sa surestimation des récompenses (*maximization bias*). De fait, Q-learning va prendre plus de risques.

SARSA

SARSA (*State–Action–Reward–State–Action*) est un algorithme d'apprentissage par renforcement créé par Rummery et Niranjan en 1994 [RN94]. Il s'inspire de Q-learning. SARSA définit la Q-fonction par :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{4.2}$$

Son fonctionnement est décrit par l'Algorithme 2. Contrairement à Q-learning, SARSA exécute deux actions pour une mise à jour. SARSA a l'avantage de prendre moins de risques que Q-learning. Néanmoins, SARSA nécessite un modèle, ce qui le rend plus difficile à utiliser que Q-learning.

Double Q-Learning

Le Double Q-learning est un algorithme d'apprentissage créé en 2010 par Hasselt [Has10]. L'intérêt du Double Q-learning est qu'il ne surestime pas les récompenses contrairement à Q-learning. Cette surestimation conduit généralement Q-learning à choisir de mauvaises actions. Pour cela, Double Q-learning utilise deux estimateurs Q : Q^A et Q^B. Similairement à Q-learning, $Q^A \approx Q^B \approx$

Algorithme 2 : Algorithme de SARSA, tel que présenté dans [SB18]

Initialiser $Q(s, a)$ pour tout état $s \in S$, toute action $a \in A$ de façon arbitraire, sauf $Q(\text{état terminal}, \cdot) = 0$;
répéter
  Initialiser S ;
  répéter
    Choisir A depuis S d'après la stratégie et Q ;
    Exécuter A ;
    Observer R et S' ;
    Choisir A' depuis S' d'après la stratégie et Q ;
    $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma Q(S', A') - Q(S, A)\right]$;
    $S \leftarrow S'$;
    $A \leftarrow A'$;
  jusqu'à S *soit l'état terminal* ;
jusqu'à *fin* ;

q_*. Ces deux estimateurs ne sont pas mis à jour en même temps, comme décrit dans l'Algorithme 3. Les résultats de Double Q-learning sont donc meilleurs que ceux de Q-learning. Le principal inconvénient reste toutefois qu'il a besoin de deux fois plus de mémoire. En effet, il est nécessaire de stocker le double des triplets action, état, récompense.

Algorithme 3 : Algorithme de Double Q-learning, tel que présenté dans [Has10]

Initialiser $Q^A(s, a)$, $Q^B(s, a)$ et s ;
répéter
  Choisir une action a d'après $Q^A(s, \cdot)$ et $Q^B(s, \cdot)$;
  Exécuter a ;
  Observer la récompense r et l'état s' ;
  Choisir entre MIS_A_JOUR(A) et MIS_A_JOUR(B) ;
  si *MIS_A_JOUR(A)* **alors**
    Définir a^* = arg max$_a Q^A(s', a)$;
    $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s, a)(r + \gamma Q^B(s', a^*) - Q^A(s, a))$;
  sinon
    Définir b^* = arg max$_a Q^B(s', a)$;
    $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s, a)(r + \gamma Q^A(s', b^*) - Q^B(s, a))$;
  fin
  $s \leftarrow s'$;
jusqu'à *fin* ;

Apprentissage par renforcement profond

Comme pour les autres paradigmes d'apprentissage automatique, il existe de l'apprentissage par renforcement profond. Toutefois, il n'apparait qu'en 2015, soit plus de 20 ans après les autres paradigmes d'apprentissage. DQN (*Deep Q-Network*) est le premier algorithme d'apprentissage par renforcement profond [Mni+15]. DQN combine Q-learning et un réseau de neurones profond. Concrètement, le réseau de neurones profond remplace la Q-fonction. Il permet notamment la gestion d'un grand nombre d'entrées, comme une image brute par exemple. Les tâches réalisables peuvent être plus complexes, mais au prix d'une complexité accrue par rapport à Q-learning. Double DQN est une variante de DQN utilisant Double Q-learning [HGS16]. Ces deux algorithmes se sont illustrés par leur utilisation dans l'intelligence artificielle AlphaGo (pour le jeu de Go) et AlphaStar (pour le jeu vidéo de stratégie en temps réel StarCraft II) [Sil+17].

4.3 Applications de l'apprentissage par renforcement aux problèmes de réseaux informatiques

Dans cette section, nous allons voir deux exemples d'application d'apprentissage par renforcement pour les réseaux informatiques. La première utilisation concerne le paramétrage du protocole TCP. La fenêtre de contention de TCP influe beaucoup sur ses performances. L'idée principale est d'utiliser l'apprentissage par renforcement pour régler ce paramètre. La seconde utilisation concerne la fonction de routage car il s'agit d'une fonction complexe. En fonction de la topologie, le plus court chemin n'est pas le meilleur. L'apprentissage par renforcement permet notamment d'utiliser des chemins alternatifs ou d'équilibrer la charge entre différents liens.

4.3.1 Application au paramétrage automatique du protocole TCP

TCP est un protocole de transport orienté connexion. Son fonctionnement est paramétrable. Un de ces paramètres est la fenêtre de contention. La gestion de cette fenêtre fait partie des problèmes complexes du domaine des réseaux. Il existe de nombreuses stratégies pour gérer cette fenêtre. Par exemple, il existe New Reno (RFC 6582), CUBIC et Vegas. Chaque stratégie définit l'incrémentation de ce paramètre ainsi que sa valeur en cas de perte de segment TCP.

Afin de résoudre le problème d'optimalité de la valeur de la fenêtre de contention, Venkata Ramana *et al.* proposent d'utiliser l'apprentissage par renforcement avec TCP. Ils nomment leur proposition Learning-TCP [VMS05]. Learning-TCP s'appuie en grande partie sur TCP Reno. Pour la partie apprentissage, Venkata Ramana *et al.* utilisent FALA (*Finite Action-set Learning Automata*). D'après leurs simulations sur *GloMoSim*, Learning-TCP permet une amélioration globale des performances, que ce soit en termes de délai de livraison ou de taux de perte. En 2011, ils proposent une nouvelle version Learning-TCP [BM11]. Cette fois, la partie apprentissage est assurée par CALA (*Continuous Action-set Learning Automata*). D'après leurs simulations sur *ns-2*, cette nouvelle version de Learning-TCP permet d'obtenir de meilleures performances en termes de taux de paquets perdus et de débit que la première version de Learning-TCP et TCP New Reno.

En 2016, Li *et al.* [Li+16] proposent TCPLearning, un variant de TCP utilisant Q-learning. Ils ont implémenté leur algorithme sur le simulateur réseau *ns-3*. Il est comparé à TCP New Reno et à d'autres variants de TCPLearning. Leur topologie de test est très simple. Leur scénario est composé d'une source, d'une destination et d'un goulot d'étranglement. Ce nœud intermédiaire autorise en fonction du scénario une bande passante soit de 7,5 Mb/s, soit de 2,5 Mb/s. Dans le premier cas, TCPLearning permet d'améliorer le débit disponible de 26 % par rapport TCP New Reno pour un RTT (*Round Trip Time*) équivalent. Dans le second cas, il offre une diminution du RTT de 12 % pour une bande passante utile équivalente. La principale faiblesse de TCPLearning reste son besoin en mémoire bien supérieur à TCP New Reno.

En 2017, Jiang *et al.* [Jia+17] proposent d'intégrer l'algorithme Q-learning dans TCP Vegas. Ils nomment leur proposition TCP-Gvegas. L'idée principale est d'utiliser Q-learning afin de déterminer la taille de la fenêtre de contention. Pour vérifier la pertinence de TCP-Gvegas, Jiang *et al.* ont implémenté leur algorithme sur le simulateur réseau *ns-2*. Ils l'ont ensuite comparé à TCP Vegas et TCP New Reno sur trois scénarios inclus avec *ns-2*. Ces scénarios ont la particularité d'utiliser des topologies sans-fil IEEE 802.11. D'après leurs simulations, TCP Gvegas permet de diminuer au moins de moitié le délai et d'augmenter d'un tiers le débit par rapport à TCP Vegas.

4.3.2 Application à la fonction de routage

Le routage fait partie des tâches les plus complexes des réseaux informatiques. Dans les années 90, deux tendances apparaissent afin de résoudre ce problème. Dans la première catégorie, il s'agit des algorithmes bio-inspirées comme les algorithmes utilisant les colonies de fourmis [Bon+98]. La deuxième approche s'inspire des algorithmes d'apprentissage par renforcement. Le premier de ces algorithmes est Q-routing [BL94]. Q-routing s'inspire de Q-learning. Nous verrons Q-routing plus en détails dans la section 4.4 qui lui est consacrée. Q-routing est le seul de ces algorithmes à avoir eu des travaux d'amélioration.

AdaR

AdaR [WW06] est un algorithme de routage conçu par Wang et Wang s'appuyant sur Q-learning. Sa particularité est d'utiliser l'ajustement des moindres carrés (*Least-Squares Policy Iteration*, LSPI). D'après leur évaluation sur une grille irrégulière de 400 capteurs, cette modification permet de diminuer le temps de convergence. Cela permet aussi d'obtenir de meilleurs résultats qu'avec Q-learning seul. D'après ses auteurs, AdaR a été conçu pour les réseaux de capteurs. Malheureusement, l'ajustement des moindres carrés augmente la complexité de l'algorithme. AdaR n'a pas été implémenté en tant que protocole de routage. Il a juste été évalué de manière analytique.

FROMS

FROMS [För+08] est un protocole de routage destiné aux réseaux de capteurs. Il a été conçu par Förster *et al.* et publié en 2008. Il utilise Q-learning afin de choisir la route. Dans leur article, ils comparent FROMS à Directed Diffusion, un protocole de routage pour les réseaux de capteurs [ST16].

D'après leur expérimentation, FROMS permet d'acheminer plus de paquets que Directed Diffusion. Son coût par paquets est aussi plus faible. Néanmoins, il est plus lent. Cependant, Directed Diffusion n'a pas été comparé à un protocole de routage standardisé. Il est donc difficile de juger de la pertinence de ces deux protocoles.

QELAR

QELAR [HF10] est un protocole de routage publié en 2010. Il est destiné aux réseaux de capteurs en environnements aquatiques. QELAR utilise Q-learning afin d'optimiser la durée de vie du réseau. Pour vérifier les performances de QELAR, Hu et Fei ont implémenté QELAR sur le simulateur *ns-2*. Ils l'ont ensuite comparé à VBF (*Vector-Based Forward*), un autre protocole de routage pour les réseaux de capteurs aquatiques. D'après leurs simulations, QELAR permet d'obtenir un taux de perte et un délai de livraison inférieur à VBF. De plus, QELAR améliore aussi la durée de vie du réseau.

4.4 Q-routing

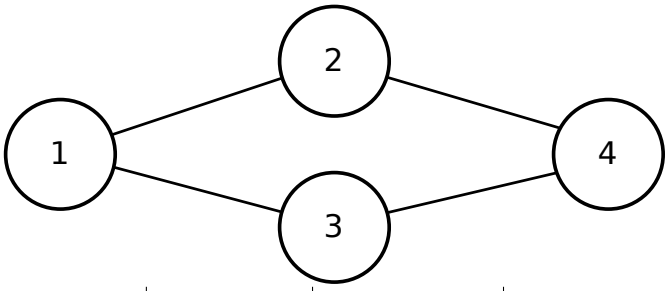
Dans cette section, nous verrons en détails Q-routing tel qu'il est décrit dans l'article de Boyan et Littman [BL94]. Nous avons choisi Q-routing car malgré sa date de publication, des travaux dérivés sont encore récemment publiés. Parmi les algorithmes spécialisés abordés à la section précédente, il est le seul avoir inspiré d'autres travaux.

4.4.1 Fonctionnement de Q-routing

Q-routing [BL94] est un algorithme de routage conçu par Boyan et Littman et publié en 1994. Il est inspiré de Q-learning [WD92], un algorithme d'apprentissage par renforcement publié deux ans plus tôt. Q-routing reprend certains concepts de Q-learning comme la Q-fonction, les Q-valeurs, mais surtout son système de mises à jour. Les Q-valeurs sont les métriques de routage de Q-routing. Elles sont calculées à partir de la Q-fonction qui associe une source x, une destination d et le prochain saut y tel que :

$$\Delta Q_x(d, y) = \eta(q + s + t - Q_x(d, y)) \tag{4.3}$$

où η est le taux d'apprentissage (généralement 0,5), q le nombre d'unités de temps passé en file d'attente, s le nombre d'unités de temps passé dans la transmission entre x et y, et t tel que : $t = \min_{z \in \text{voisins de } y} Q_y(d, z)$. Une route est donc définie à minima par son origine, sa destination, le prochain saut ainsi que la Q-valeur associée. Il peut donc y avoir plusieurs routes disponibles pour une même destination. La route effectivement utilisée est la route ayant la plus petite Q-valeur. La Figure 4.3 illustre les routes sur le nœud 1 et à destination du nœud 4. Sur cet exemple, la route vers le nœud 4 passant par le nœud 2 serait sélectionnée. Q-routing met à jour uniquement les Q-valeurs des routes sélectionnées. Sur l'exemple précédent, seule la route vers 4 passant par 2 aura sa Q-valeur actualisée. À la suite de l'actualisation d'une route, Q-routing choisit à nouveau la route avec la plus petite Q-valeur.



Source	Destination	Prochain saut	Q-valeur
1	2	2	10
1	3	3	10
1	4	2	20
1	4	3	20
1	2	3	30
1	3	2	30

FIG. 4.3 : Exemple de routes présentes sur le nœud 1 et à destination de 4 après exploration.

4.4.2 Analyse des résultats obtenus par Boyan et Littman

Dans leur article, Boyan et Littman ont évalué Q-routing sur plusieurs topologies réseaux dont leur grille irrégulière 6×6. La Figure 4.4 représente cette topologie réseau. Elle est composée de 36 nœuds répartis sur deux sous-grilles reliées par deux chemins de longueur différente. Boyan et Littman ont publié uniquement les résultats pour cette topologie particulière.

Q-routing est comparé à l'algorithme de Bellman-Ford, un algorithme de plus court chemin. D'après leur simulation, Bellman-Ford offre un meilleur délai de bout à bout à faible charge. En effet, à faible charge, Q-routing n'est pas toujours capable de trouver le chemin le plus rapide. À plus forte charge, Q-routing équilibre la charge entre deux chemins. Cela lui permet d'éviter une congestion sur le lien entre les nœuds 16 et 22. Q-routing arrive ainsi à maintenir un délai de livraison plus faible que Bellman-Ford. Cependant, lorsque les deux chemins reliant les deux sous-grilles sont saturés, le délai de livraison augmente fortement. Malheureusement, le taux de livraison des paquets n'est pas indiqué. Cette métrique est importante car Q-routing pourrait détruire des paquets afin que les paquets restants puissent être acheminés plus rapidement.

Les travaux de Boyan et Littman sont très difficilement reproductibles. Les simulations ont été effectuées sur leur simulateur développé en interne. Certaines simplifications sont présentes comme les files d'attente de taille illimitée. Les hypothèses de fonctionnement du simulateur ne sont pas toutes connues. Par exemple, bien que Q-routing soit distribuable, il n'est pas précisé s'il fonctionnait de façon distribuée sur leur simulateur. Chaque niveau de charge n'est pas défini. De plus, la répartition des couples sources-destinations n'est pas spécifiée. Sur une grille irrégulière comme celle-ci, cette répartition a une grande importance pour estimer les chemins utilisés. Enfin, Q-routing a

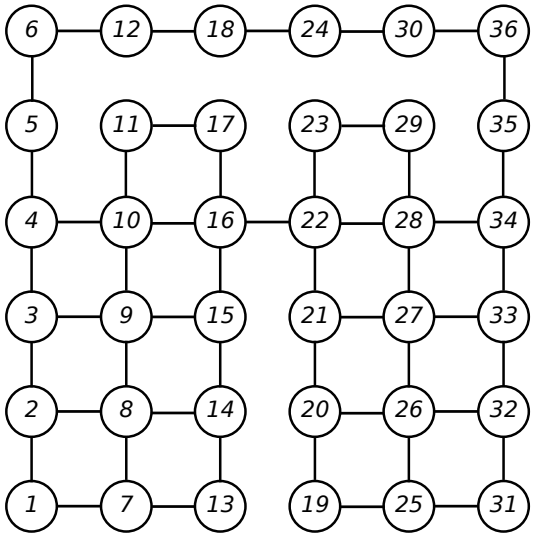


FIG. 4.4 : Grille irrégulière 6×6 de Boyan et Littman, extrait de [BL94].

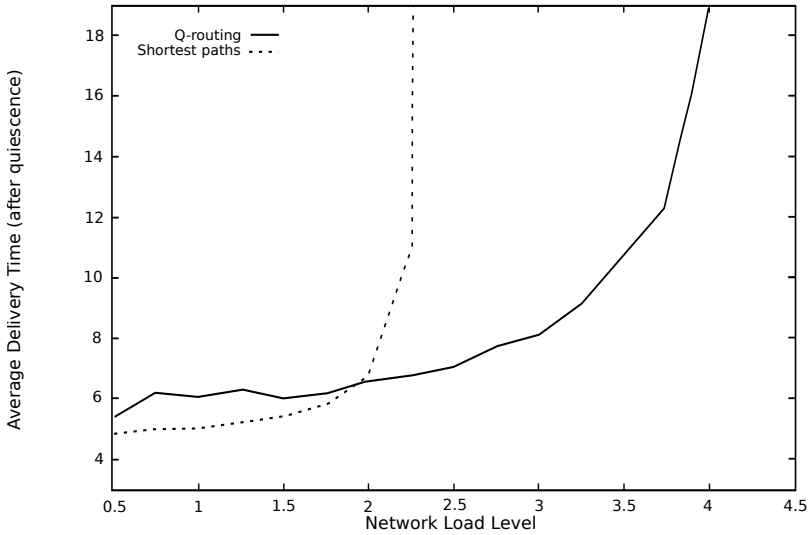


FIG. 4.5 : Délai de livraison en fonction de la charge réseau pour Q-routing et Bellman-Ford, extrait [BL94].

un comportement glouton à cause de sa stratégie de choix et de son système de mise à jour. Ce comportement se traduit par une sensibilité aux optimums locaux. Concrètement, une congestion même momentanée peut priver Q-routing de sa meilleure route.

4.5 Modifications génériques de Q-routing

Quelques années plus tard, plusieurs modifications ont été proposées pour combler les faiblesses de Q-routing. Predictive Q-routing est le premier d'entre eux. Son objectif est de corriger deux faiblesses. La première est le comportement glouton dû à la stratégie de Q-routing. La seconde faiblesse concerne les performances de Q-routing à faible charge.

4.5.1 Predictive Q-routing

En 1996, Choi et Yeung [CY96] proposent une amélioration de Q-routing. Ils l'appellent Predictive Q-routing (PQ-routing). PQ-routing a vocation de combler deux faiblesses. Premièrement, Q-routing a des difficultés à trouver le chemin optimal lorsque la charge réseau est faible. Choi et Yeung proposent d'utiliser un algorithme de plus court chemin lorsque la charge réseau est faible. Les effets de la stratégie gloutonne sont la seconde faiblesse que Choi et Yeung souhaitent atténuer. Pour cela, PQ-routing utilise 4 tables différentes : Q (comme Q-routing), B (meilleures Q-valeurs), R (taux de récupération des routes) et U (horodatage de la dernière mise à jour). Avec ces quatre tables, PQ-routing peut restaurer les anciennes Q-valeurs après une congestion. Contrairement à Q-routing, le taux d'apprentissage de PQ-routing est de 0,7. Choi et Yeung comparent PQ-routing à Q-routing sur deux topologies : une topologie originale de 15 nœuds et la grille irrégulière 6×6 de Boyan et Littman. L'évaluation porte uniquement sur le délai de bout-en-bout. PQ-routing offre un meilleur délai de bout-en-bout sur les différents scénarios incluant une variation du trafic réseau. Dans le cas où le trafic est important et apériodique, les résultats sont moins favorables pour PQ-routing. La durée d'apprentissage est aussi réduite grâce à PQ-routing. Cependant, comme Boyan et Littman, Choi et Yeung ont évalué leur algorithme sur leur simulateur développé en interne. Bien que certains paramètres de simulation soient donnés, l'emplacement des couples source-destination ne sont pas précisés pour la grille irrégulière.

4.5.2 Dual Reinforcement Q-Routing

Dual Reinforcement Q-routing (DRQ-routing) [Kum97] est un algorithme de routage dérivé de Q-routing. DRQ-routing a été conçu par Kumar et publié en 1997. Il apporte la rétropropagation des informations de la destination vers la source par rapport à Q-routing. Cette idée est issue de l'apprentissage par renforcement dual de Goetz *et al.* [GKM96]. Le taux d'apprentissage lors de la propagation est 0,7 et 0,9 lors de la rétropropagation. Le taux d'apprentissage de la rétropropagation est plus élevé, car la rétropropagation est considérée comme plus fiable que la propagation. Afin de valider la pertinence de la rétropropagation, Kumar compare DRQ-routing à Q-routing et l'algorithme de plus court chemin de Bellman-Ford. Il évalue ces trois algorithmes sur la grille irrégulière 6×6 de Boyan et Littman. L'évaluation porte uniquement sur le délai de livraison. Comme les travaux précédents, Kumar utilise leur simulateur développé en interne. Certaines hypothèses ne sont pas réalistes. Par exemple, la taille des files d'attente n'est pas limitée. D'après les simulations, DRQ-routing permet un apprentissage plus rapide à forte charge. DRQ-routing délivre les paquets

plus rapidement que Q-routing indépendamment de la charge réseau. Il reste un peu plus lent que Bellman-Ford à faible charge. Kumar estime que le surcoût lié à la rétropropagation est négligeable.

4.5.3 Q-Neural Routing

Q-Neural Routing est un algorithme dérivé de Q-routing conçu par Said Hoceini au cours de sa thèse de doctorat [Hoc04]. Sa particularité est d'utiliser un réseau de neurones à la place de la Q-fonction. Le réseau de neurones reçoit en entrée la destination et renvoie le prochain saut. D'après les simulations réalisées sur *OPNET*, Q-Neural Routing permet d'obtenir un délai de livraison plus court qu'avec Q-routing à moyenne et forte charge. Néanmoins, Q-Neural Routing a besoin de certaines hypothèses pour fonctionner. Par exemple, il est nécessaire de connaitre au préalable le nombre de destinations et de prochains sauts afin de pouvoir dimensionner le réseau de neurones. Cela implique aussi que la topologie soit statique. Le temps de convergence est aussi plus long avec Q-Neural Routing que Q-routing à cause de l'entraînement du réseau de neurones.

4.5.4 K-Shortest path Q-routing

K-Shortest path Q-routing est un algorithme hybride issue de la thèse de doctorat de Said Hoceini [Hoc04]. Il combine à la fois l'algorithme de Dijkstra généralisé et Q-routing. L'algorithme de Dijkstra permet de déterminer k plus courts chemins. Q-routing sélectionne un des k plus courts chemins. L'espace de recherche de K-Shortest path Q-routing est plus restreint que celui de la version originale de Q-routing car il se limite à k chemins. Il y a néanmoins au moins deux contreparties. D'après Hoceini, un mécanisme de détection des boucles de routage est nécessaire. De plus, il est nécessaire de pouvoir détecter les changements de topologies afin de recalculer les k plus courts chemins. Hoceini a implémenté Q-routing et son dérivé sur *OPNET*. Il a ensuite évalué K-Shortest path Q-routing sur deux topologies inspirées de la grille irrégulière de Boyan et Littman. D'après ses simulations, K-Shortest path Q-routing offre un meilleur délai de bout-en-bout que Q-routing et RIP, un protocole standardisé (RFC 1058) et implémentant l'algorithme de Bellman-Ford. Néanmoins, aucune autre métrique comme le taux de paquets livrés n'est fournie.

4.5.5 Q^2-routing

Q-routing ne traite pas les demandes particulières en termes de qualité de services. Par exemple, il n'est pas possible de paramétrer Q-routing pour optimiser la gigue. En 2018, Hendriks *et al.* [HCL18] proposent de remédier à ce besoin. Pour cela, ils créent Q^2-routing, un algorithme dérivé de Q-routing. Ils proposent d'ajouter trois coefficients C_d (délai), C_j (gigue) et C_l (taux de paquets livrés). Ces coefficients sont paramétrés en fonction des exigences en qualité de services. Il modifie la Q-fonction tel que :

$$Q_x(y,d) = (C_d \times C_l \times C_j) \times ((1-\alpha) \times Q_x(y,d) + \alpha(q+s+t)) \tag{4.4}$$

Hendriks *et al.* utilisent un taux d'apprentissage $\alpha = 0,5$. Ils proposent aussi l'utilisation d'une stratégie ε-glouton pour le choix des routes lors de la phase d'apprentissage. Enfin, Hendriks *et al.*

assurent que Q^2-routing est un protocole de routage hybride, complètement distribué et ne génère pas de boucle de routage. Afin de vérifier le fonctionnement et la pertinence de leurs modifications, Hendriks *et al.* ont implémenté Q^2-routing et Q-routing sur le simulateur réseau *ns-3*. Ils ont pu ainsi comparer leur implémentation de Q^2-routing à leur implémentation de Q-routing et AODV (RFC 3561), un protocole de routage ad-hoc. Ils proposent un scénario avec une topologie sans-fil avec trois chemins entre la source et la destination. Différentes perturbations apparaissent sur les trois chemins en cours de la simulation. Hendriks *et al.* utilisent le délai et le pourcentage de paquets conformes à une classe de trafic pour évaluer ces trois protocoles. D'après leurs simulations, Q^2-routing obtient dans l'ensemble de meilleurs résultats qu'AODV. Cependant, ces résultats ont été obtenus au prix d'un surcoût en termes de charge réseau.

4.6 Adaptation de Q-routing à des scénarios particuliers

Dans cette section, nous abordons trois adaptations de Q-routing pour trois scénarios particuliers : *i)* la mobilité, *ii)* les réseaux de capteurs et *iii)* la radio cognitive. Q-routing a été conçu pour les réseaux filaires. Ces scénarios n'ont donc pu être traités par Boyan et Littman.

4.6.1 Q-routing sur des scénarios avec mobilité

Q-routing a été créé avant IEEE 802.11. Il n'a pas été conçu pour les réseaux sans-fil et encore moins pour des scénarios avec mobilité. En 2020, Serhani *et al.* [SNJ20] entendent résoudre le cas de la mobilité avec Adaptive Q-routing (AQ-routing). AQ-routing est un algorithme dérivé de Q-routing. Il est spécialisé dans les scénarios avec mobilité. AQ-routing utilise une Q-fonction complexe s'appuyant sur la stabilité plutôt que la latence :

$$Q_{metric_{ij}} = \alpha_{ij} \cdot \varphi(MF_j) + (1 - \alpha_{ij}) \cdot \lambda ETX_{ij} \tag{4.5}$$

où i est un nœud, j un nœud voisin de i, MF est le facteur de mobilité, α_{ij} le taux d'apprentissage, γ un facteur de normalisation pour la fonction ETX et $\varphi(MF_j)$ est défini tel que :

$$\varphi(MF_j) = \frac{a}{1 - e^{\frac{-MF_j}{b}}} \tag{4.6}$$

Ils utilisent un taux d'apprentissage $\alpha = 0,95$ pour leurs simulations. Dans leur article, ils comparent AQ-routing à OLSR (*Optimized Link State Routing*, standard et ETX) sur le simulateur *ns-3*. Afin d'obtenir des performances optimales, leur implémentation d'AQ-routing s'inspire de certains concepts d'OLSR, un protocole de routage standardisé et spécialisé pour les réseaux MANETs (*Mobile Ad-hoc NETworks*). Par exemple, leur implémentation reprend les paquets HELLO ainsi que les paquets TC (Topology Control). Serhani *et al.* évaluent AQ-routing sur un cas statique et un cas avec mobilité. Le taux de paquets livrés et le délai de livraison sont les métriques d'évaluation. La topologie est composée de 30 nœuds répartis aléatoirement sur espace de 1000×1000 m. Dans le cas

statique, AQ-routing offre un taux de paquets livrés quasi-identique à OLSR-ETX. Cependant, AQ-routing achemine en moyenne les paquets plus lentement que les deux variants d'OLSR, soit 205 ms contre 173 ms pour OLSR-ETX. À basse vitesse (entre 0 et 3 m/s), OLSR-ETX offre les meilleures performances. Le taux de paquets livrés avec OLSR-ETX est équivalent à celui d'AQ-routing. OLSR-ETX livre aussi les paquets plus rapidement. À partir d'une mobilité 3 m/s, les résultats sont favorables à AQ-routing que ce soient en termes de taux de paquets livrés qu'en délai de livraison. Cependant, Serhani *et al.* ont augmenté la complexité de la Q-fonction utilisant la métrique ETX ainsi que d'autres coefficients pour obtenir ces résultats.

4.6.2 Q-routing pour les réseaux de capteurs

Q-routing n'est pas conçu spécifiquement pour les réseaux sans-fil de capteurs (*Wireless Sensor Network*, WSN). Bouzid *et al.* [Bou+20] proposent R2LTO (*Reinforcement Learning for LT Optimisation*), une version de Q-routing adaptée aux réseaux de capteurs. Leur changement porte essentiellement sur la récompense. Ils utilisent l'énergie pondérée par le nombre de sauts. Ils ont comparé leur algorithme à Q-routing et RLBR, un autre algorithme spécialisé, sur leur simulateur. D'après leur simulation, R2LTO améliore la durée de vie du réseau jusqu'à 25 %. Les autres métriques de qualité de service ne sont pas présentées. De plus, les simulations portent uniquement sur l'algorithme et non sur une éventuelle implémentation protocolaire. Le surcoût lié à la gestion du routage n'est donc pas évalué.

4.6.3 Q-routing pour la radio cognitive

La radio cognitive s'appuie sur la radio logicielle soit pour optimiser les couches physiques, voire liaisons des réseaux sans-fil, soit pour réutiliser certaines bandes de fréquences sous-utilisées hors ISM. Dans le deuxième cas, le routage peut s'avérer délicat car le réseau peut changer fréquemment de bandes de fréquences ou être sur plusieurs bandes de fréquences. Xia *et al.* [Xia+09] proposent d'adapter Q-routing et DRQ-routing pour la radio cognitive. Afin d'évaluer les performances de leurs modifications, ils les ont implémentés sur le simulateur *OMNet++*. Ils sont comparés à un protocole du plus court chemin optimisé pour la radio cognitive. D'après leurs simulations, Q-routing et DRQ-routing offrent en moyenne un délai de livraison plus court que le protocole type plus court chemin. DRQ-routing offre en moyenne le meilleur délai de livraison. Son temps de convergence est aussi plus court de que celui de Q-routing.

4.7 Conclusions

Dans ce chapitre, nous avons abordés quelques algorithmes d'apprentissage par renforcement. Parmi eux, Q-learning est un algorithme qui offre un bon rapport entre efficacité et simplicité de mise en œuvre. Par exemple, trois des quatre algorithmes de routages présentés utilisent Q-learning ou s'en inspirent. En effet, Q-learning ne nécessite ni modèle, ni jeu de données initial. Parmi tous les travaux cités, nous nous sommes concentrés sur l'algorithme de routage Q-routing. Q-routing offre

des résultats prometteurs sur le simulateur de Boyan et Littman [BL94]. Au fil des années, de nombreuses modifications sont apparues comme Predictive Q-routing [CY96] ou Q^2-routing [HCL18]. Nous avons vu que certaines modifient les fonctionnements de Q-routing comme Q-Neural Routing [Hoc04]. D'autres modifications sont destinées à adapter Q-routing à une situation particulière comme la radio cognitive [Xia+09]. Dans la prochaine partie, nous aborderons nos contributions : l'implémentation et les améliorations Q-routing.