Modélisation et simulation DEVS des effets de levier financier en apprentissage par renforcement

E. Barbieri[1*] L. Capocchi[1*] J.F. Santucci[1*]
[1] University of Corsica
SPE UMR CNRS 6134
Campus Grimaldi, 20250 Corte
*{barbieri_e, capocchi_l, santucci_j}@univ-corse.fr

Résumé :

La prise de décision au cours d'un processus d'optimisation des actifs financiers menant à un éventuel effet de levier est un enjeu majeur durant la phase d'instruction des dossiers des porteurs de projet dans le cadre d'un programme d'investissement tel que les programmes de développement Européen. La modélisation et la simulation basées sur un apprentissage par renforcement peuvent permettre de proposer une politique de décision optimale dans ce genre de processus. Ce papier présente une approche de modélisation et de simulation à événements discrets DEVS à partir des processus décisionnels Markovien appliquée à la recherche des effets de levier maximum sur les capacités d'autofinancement en phase d'instruction de dossier de subvention. La mise en application de l'approche présentée dans ce papier est faite sur la recherche de l'effet levier lié à la volatilité des cours des principaux indices boursier (CAC40, NasDaq, etc.)

Mots-clés :

modélisation, simulation, DEVS, DEVSimPy, Markov, effet de levier, décision.

Abstract:

Decision making during a financial asset optimization process leading to a potential leverage effect is a major issue in the management of an investment program such as European development programs. Modeling and simulation based on reinforcement learning can propose a decision-making policy in this kind of process. This paper presents a DEVS discrete-event modeling and simulation approach from Markov decision-making processes applied to the search for maximum leverage on self-financing capabilities in grant application instruction phase. The application of the approach presented in this paper is made on the search for the leverage effect linked to the price volatility of the main stock market indices (CAC40, NasDaq, etc.).

1 Introduction

Du fait du nombre important d'états et d'actions à considérer dans la gestion des processus financiers afin de réaliser un éventuel effet de levier, leur conduite fait souvent appel à la simulation. A contrario, l'évaluation ex-ante des effets de levier d'un plan de développement n'exploite pas les avantages liés à l'utilisation de la simulation dans le cadre de l'apprentissage par renforcement. Généralement les méthodes utilisées sont des méthodes faisant référence à des modèles de comptabilité d'équilibre général qui permettent d'évaluer les chocs produits par des choix politiques [1]. Les modèles de comptabilité en équilibre général nécessitent généralement de s'affranchir de la définition d'une matrice de probabilité car ils ne permettent pas de définir des politiques décisionnelles à large échelle.

Un processus d'optimisation des actifs financiers répond à la propriété des chaînes de Markov qui considère que l'état suivant d'un système ne dépend que de l'état présent. Autrement dit, l'historique des transitions passées n'influence pas la détermination du futur état. Si le processus d'optimisation est celui de la bourse, la propriété de Markov est clairement identifiée par le fait que les performances passées ne préjugent pas des performances futures. L'approche consistant à avoir recours à la modélisation sous la forme d'un Processus Décisionnel Markovien (PDM) pour représenter un processus décisionnel appliqué à la gestion des actifs semble être la plus appropriée à la nature même particulièrement volatile des marchés financiers. De plus, l'approche Markovienne basée simulation [2] correspond à notre contexte de simulation dans lequel la prise de décision doit être conduite avec un fort degré d'incertitude sans la possibilité de définir à priori un cadre de transition des états.

La définition d'un modèle de PDM nécessite un cadre formel permettant la spécification modulaire afin de représenter l'interaction (actions/récompenses) entre l'environnement et l'agent d'un PDM. Le formalisme DEVS (Discrete EVent system Specification) [3], est le cadre mathématique approprié permettant de spécifier de manière formelle un PDM. De plus, DEVS offre la simulation automatique de ces modèles et donc la simulation des algorithmes d'apprentissage par renforcement (par exemple le Q-Learning) [4] appliquée aux PDM dans le but d'obtenir la solution optimale dans le cadre d'un problème décisionnel soumis à un fort degré d'incertitude. Dans [5], les auteurs proposent d'utiliser DEVS comme cadre expérimental pour la mise en oeuvre de l'intégration du temps dans les PDM afin de définir des dates de décision. Cette approche concorde avec notre besoin de traiter le temps comme une donnée fondamentale dans notre processus décisionnel.

Dans ce papier nous présentons une modélisation d'un processus d'optimisation des actifs financiers afin de définir la politique optimale à appliquer pour effectuer un effet de levier sur les capacités d'autofinancement des acteurs économiques. Cette modélisation est réalisée à l'aide du formalisme DEVS [3] et des processus décisionnels Markoviens résolus via simulation basée sur de l'apprentissage par renforcement et plus précisément à l'aide de l'algorithme Q-Learning.

Le papier est organisé de la manière suivante : après avoir présenté comment l'apprentissage par renforcement et la simulation peuvent être intégré et comment le formalisme DEVS peut venir en aide dans cette intégration, nous présentons dans une seconde partie la problématique d'obtention des effets de levier dans les processus d'optimisation des actifs financiers. Une troisième partie présente le modèle mathématique du PDM utilisé par la suite dans un cadre défini par DEVS. Avant de conclure et de donner des perspectives, un exemple sur la recherche de l'effet levier lié à la volatilité des cours des principaux indices boursier (CAC40, NasDaq, etc.) est proposé dans l'environnement DEVSimPy

2 Apprentissage par renforcement et simulation DEVS

Les systèmes qui utilisent déjà l'IA (Intelligence Artificielle), tels que les chaînes d'approvisionnement numérique, les "usines intelligentes" et d'autres processus industriels de l'industrie 4.0, devront inévitablement inclure l'IA dans leurs modèles de simulation. Par exemple, avec les systèmes d'analyse de simulation, les composants IA peuvent être directement intégrés dans le modèle de simulation pour permettre les tests et les prévisions. Dans [6] les auteurs utilisent un algorithme d'apprentissage par renfoncement (Q-Learning) afin de combiner un algorithme d'équilibrage de charge dynamique et un algorithme à fenêtre bornée pour la simulation à événements discrets des circuits VLSI (Very-large-scale integration) au niveau porte.

L'application de l'IA à l'optimisation est une autre opportunité d'intégration dans la modélisation de simulation. Les systèmes basés sur des agents ont souvent beaucoup de paramètres et nécessitent des temps d'exécution importants pour explorer toutes leurs permutations afin de trouver la meilleure configuration des modèles. L'IA peut accélérer cette phase de configuration et fournir une optimisation plus efficace. En retour, la simulation peut également accélérer les processus d'apprentissage et de configuration des algorithmes de l'IA. En effet dans [7], la simulation DEVS comparative et concurrente est utilisée pour tester toutes les configurations possibles des hyper-paramètres (momentum, learning rate, etc) d'un réseau de neurones.

Dans le cas de l'apprentissage par renforcement, des composants peuvent être développés pour remplacer les modèles basés sur des règles.

Ceci est possible lorsque l'on considère le comportement humain et la prise de décision. Par exemple, dans [8] les auteurs considèrent qu'en observant un comportement désiré, sous la forme de sorties produites en réponse à des entrées, un modèle DEVS comportemental équivalent peut être construit. Ces composants d'apprentissage peuvent soit être utilisés dans des modèles de simulation pour refléter le système réel, soit être utilisés pour former les composants d'IA. En générant les ensembles de données nécessaires à l'apprentissage des réseaux de neurones, les modèles de simulation peuvent être un outil puissant lors du déploiement des algorithmes de l'apprentissage par renfoncement.

Les techniques d'apprentissage de l'IA ont déjà été utilisées dans un contexte de simulation DEVS. En effet, dans [9] les auteurs proposent l'intégration de certains algorithmes prédictifs d'apprentissage automatique dans le simulateur DEVS afin de réduire considérablement les temps d'exécution de la simulation pour de nombreuses applications sans en compromettre la précision.

De manière plus générale, le formalisme DEVS peut être utilisé pour faciliter le développement des trois phases traditionnelles intervenant dans un processus d'apprentissage par renforcement d'un système donné sur la figure 1 :

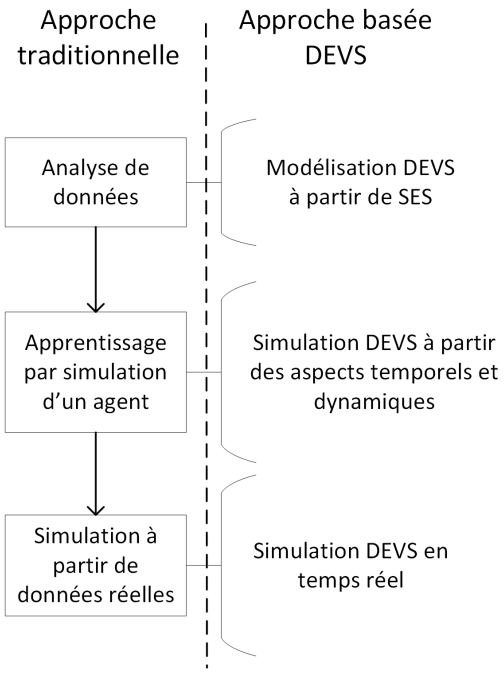

Figure 1 – Apprentissage par renforcement à partir de DEVS.

— L'analyse de données consiste en une analyse exploratoire des données qui va permettre de déterminer le type d'algorithme apprentissage (supervisé, non supervisé, par renforcement) à mettre pour un problème décisionnel donné. De plus, cette phase permet également de déterminer les variables d'état du futur modèle d'apprentissage. Cette phase est l'une des plus importante dans le processus de modélisation. Comme il est noté sur la figure 1, le formalisme SES (System entity Structure) [10] peut être utilisé pour définir une famille de modèles d'algorithmes d'apprentissage (DQN, DDQN, A3C, Q-Learning, etc.) [11] en fonction du résultat de l'analyse des données basées sur l'interprétation de statistiques.
— L'apprentissage par simulation d'un agent consiste à simuler des jeux d'entrées du modèle d'apprentissage pour calibrer celui-ci en évitant le sur-apprentissage (phase d'apprentissage). Le formalisme DEVS permet de simuler l'environnement en tant que modèle atomique. Cependant, l'environnement qui est en interaction avec l'agent dans le cadre d'une schéma traditionnel d'apprentissage par renforcement, peut également être considéré comme in modèle couplé composé de plusieurs modèles atomiques interconnnectés. C'est dans ce cadre que l'on considère l'environnement comme un modèle DEVS dynamique multi-agent dans lequel le nombre d'agent peut varier dans le temps.

— La simulation en temps réel consiste à soumettre le modèle aux données d'entrée réelles (phase de test). Le formalisme DEVS et son cadre expérimental et un excellent candidat pour simuler les politiques de décision à partir de données de simulation réelles.

Dans ce papier nous proposons de nous appuyer sur le cadre formel proposé par le formalisme DEVS afin de modéliser par la simulation, le processus d'effet levier à partir de l'algorithme d'apprentissage Q-Learning. Le formalisme DEVS permet d'isoler le Q-Learning dans un modèle atomique réagissant par sa fonction de transition externe à chaque changement significatif d'un certain nombre d'index financier.

3 Les effets de levier dans les processus d'optimisation des actifs financiers

Il s'agit de la possibilité d'emprunter un capital en apportant la capacité d'autofinancement (CAF) en couverture du risque de perte d'une partie du capitale emprunté. Ainsi, la volatilité des valeurs boursières s'applique non pas à la valeur nominale de la CAF mais à un capital emprunté de 2 à 10 fois supérieur, ce qui permet d'espérer une augmentation de la CAF finale qui servira au calcul de l'effet levier dans le cadre d'un co-financement public. Imaginons l'investissement d'une CAF de 40 000 euro (40k/euro) en optant pour un panier d'investissement (équivalent à un état) constitué de 1 à 3 indices boursiers ($index_a, index_b, index_c$) parmi les principaux indices mondiaux (CAC40, Nasdaq, etc.). Le premier effet levier consiste à obtenir une valeur de cet état v de 2 à 10 fois supérieur à la CAF initiale ($2 * caf < v < 10 * caf$). Ainsi nous avons un nouvel état "effet levier". L'objectif est d'avoir un panier d'une multiplicité (nombre de titre) de 0 à 4 de chacun de ces indices qui nous permet à tout instant d'obtenir la valeur maximale possible en additionnant l'ensemble des valeurs des indices pondéré par le nombre de titre effectivement détenu :

$$max \sum_{i=a,b,c} \{0-4\} * index_i \qquad (1)$$

Afin d'éviter toutes pertes, un état "2%" garanti annuel (250 jours) servira d'état d'attente pour accueillir le capital investi dès lors que le résultat de l'équation 1 est inférieur ou égal à $0.02/250.0$.

Investir la CAF en indices boursiers plutôt qu'en actions individuelles d'entreprises permet de prendre en compte en partie l'évolution de l'environnement des marchés financiers pour définir notre politique. En effet, la volatilité des indices (qui représente la tendances des meilleurs actions du marchés) reflète de facto le comportement des agents majeurs qui influent sur les tendances des marchés et donc sur son environnement (correction, haussier, baissier, etc.). Même en limitant la représentation de l'environnement à la seule volatilité des indices ou a un nouvel apport de trésorerie (cash) à placer en titre d'indices boursiers, il est important de remarquer à quel point l'environnement peut avoir un impact sur notre simulation. En effet, le changement de valeur d'un indice ou la disponibilité de nouveau cash influencera, par exemple le nombre d'état ou leur durée de vie. Notre exemple de simulation traitera justement de la politique a mener dans le cas d'une variation d'environnement liée à l'augmentation de cash disponible a devoir investir.

Il existe actuellement deux principales méthodes de prise de décision sur les marchés : le trading à haute fréquence qui utilise les signaux micro structuraux des marchés et l'approche plus moyen long terme qui se base sur une prise en compte de facteurs sur le long terme comme par exemple les résultats prévisionnels des entreprises, des données macro-économiques ou encore des facteurs de géopolitique. En lien avec l'exigence d'optimiser la CAF en vue d'un investissement

dans le cadre d'un PDR, horizon moyen 1 an, et compte tenu du fait qu'il serait difficile de rentrer en compétions avec le trading en haute fréquence qui utilise des canaux dédiés pour effectuer les transactions, nous lancerons une simulation toutes les heures pour espérer être plus rapide qu'un opérateur humain qui travaille avec ses balises et plus performant grâce à l'apprentissage que les algorithmes à haute fréquence.

4 Modélisation par processus décisionnel Markovien basé simulation

Les processus de décision de Markov (PDM) basés simulation sont définis comme des processus stochastiques contrôlés satisfaisant la propriété de Markov, assignant des récompenses aux transitions d'état [12, 13]. Un PDM peut être défini avec l'ensemble (S, A, T, r) où :

- S est l'espace d'état dans lequel le processus évolue.
- A est l'espace des actions qui contrôle la dynamique de l'état.
- T est l'espace temporel ou l'axe temporel.
- r() est la fonction de récompense sur les transitions entre états.

Dans le cas d'un PDM à large échelle, basé simulation, l'équation de Bellman[14] nous permet de déterminer une politique optimale sans générer une matrice de transition de probabilités comme c'est le cas dans l'approche traditionnelle de résolution de prise de décision par PDM. L'équation de Bellman (équation 2) stipule que la récompense à long terme attendue pour une action donnée est égale à la récompense immédiate de l'action actuelle combinée avec la récompense attendue de la meilleure action future prise dans l'état suivant. Cela signifie que la valeur Q pour un ou plusieurs états et l'action (a) devraient représenter la récompense actuelle (r) plus la récompense future maximale escomptée (γ) attendue pour l'état suivant (s'). Le facteur γ permet d'apprécier à plus ($\gamma = 1$) ou moyen terme ($\gamma < 1$) les futures valeurs de Q.

$$Q(s, a) = r + \gamma(max(Q(s', a')) \qquad (2)$$

avec $s' \in S$ et $a' \in A$.

La variable Q nous permet de décider de l'importance des récompenses futures possibles par rapport à la récompense actuelle. En effet, c'est une matrice composée d'un nombre de lignes équivalent au nombre d'états et d'un nombre de colonnes équivalent au nombre d'actions considérées. L'algorithme du Q-Learning est utilisé pour déterminer, par itération, la valeur optimale de la variable Q.

5 Étude de cas : effet de levier sur indice boursier

Nous choisissons l'environnement de modélisation et de simulation DEVSimPy [15] pour implémenter en langage Python l'exemple d'une CAF investie dans 3 indices boursiers : CAC40, NasDaq et DAX. La figure 2 présente le modèle DEVS avec les modèles atomiques suivants :

- *CAF* : un générateur de la CAF prise en compte dans le processus décisionnel auquel est appliqué un effet levier de 2 à 10.
- *Index_CAC40* : un générateur de l'indice CAC40.
- *Index_NasDaq* : un générateur de l'indice NasDaq.
- *Index_DAX* : un générateur de l'indice DAX.
- *States* : un modèle qui génère tous les états possibles (pouvant aller jusqu'à toutes les combinaisons possibles en prenant un nombre de titre de 0 à 4 pour tous les indices)
- *Actions* : un modèle qui génère toutes

les actions possibles sur la base de la listes des états générés par le modèles States.

— *Rewards* : un modèle qui définit les récompenses pour chaque action générée par le modèle Actions.

— *Q-Learning* : un modèle qui exécute l'algorithme du Q-Learning à partir de l'exécution de sa fonction de transition interne. Ce modèle est préempté par le modèle States sur la base des fréquences d'acquisition des modèles Index.

— *Politique_optimal* : un collecteur qui affiche la politique optimale obtenue.

5.1 Modélisation DEVSimPy

La figure 2 détaille la modélisation sous l'environnement DEVSimPy de l'exemple présenté dans cette section.

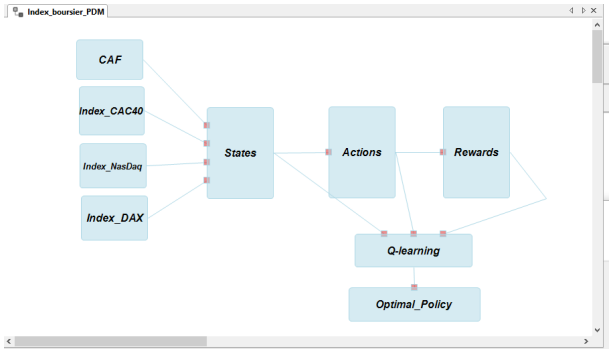

Figure 2 – Modélisation DEVSimPy.

Le listing 1 présente la fonction de transition externe du modèle Q-Learning dans laquelle la variable self.Q (ligne 27) permet de déterminer la politique à appliquer. Les propriétés s et self.num_episodes (lignes 5 et 4) sont l'état initial et le nombre d'épisode nécessaire à l'apprentissage. La propriété self.A est la matrice représentant l'automate d'états. Les propriétés self.actions et self.possible_state (ligne 25) permettent de stocker les actions et les états à considérer en fonction de l'ifc simulé. Les propriétés self.lr (learning rate) et self.y (ligne 27) sont des paramètres relatifs à l'algorithme du Q-Learning. Enfin la méthode self.step(s,a) (ligne 23) permet d'obtenir un nouvel état (s1), une récompense (r1) et un booléen indiquant si l'état final est atteint (d) en fonction d'un état courant s et d'une action à appliquer a. La variable cash (ligne 18) permet de maîtriser l'argent disponible et donc rend possible l'évolution de l'état courant. Si le cache est négatif (ligne 20), le future état s1 et l'état courant s et il n'y aucune récompense à attribuer. Nous reviendrons sur cette modification de l'algorithme du Q-Learning dans l'interprétation des résultats de la sous-section Simulation.

```
1
2  def function extTransition(self):
3   rList = []
4   for i in range(self.num_episodes):
5    s = self.init_state
6    rAll = 0
7    d = False
8    j = 0
9    cash = init_cash
10
11   while not d:
12    j+=1
13    n = np.random.randint(0,len(self.A
       [str(s)]))
14    a = self.A[str(s)][n]['action']
15
16    if a not in ['wait','go_to_2pct']:
17     value = filter(lambda a: a<0 or a
        >0, a)[0]
18     cash += -int(value)*self.
       index_dict[self.index_list[a.index
       (value)]]['value']
19
20    if cash<0: s1,r,d,_=(s,0,False,"")
21    else: s1,r,d,_=self.step(s, a)
22
23    k,l=self.possible_states.index(s),
       self.actions.index(a)
24
25    self.Q[k,l] = self.Q[k,l] + self.
       lr*(self.r + self.y*np.max(self.Q[
       self.possible_states.index(s1),:])
        - self.Q[k,l])
26    rAll += r
27    s = s1
28   rList.append(rAll)
```

Listing 1 – Code python de la fonction de

transition externe du modèle Q-Learning

L'algorithme présenté dans le listing 1 consiste à itérer (ligne 4) une portion de code conditionné par un "tant que" (ligne 11) qui à pour objectif de calculer la récompense cumulée (ligne 27) jusqu'à obtention de l'état final (d à ligne 11).

5.2 Simulation DEVSimPy

Après la simulation du scénario suivant :

- CAF : 20k€.
- Index_CAC40 : 6k€.
- Index_NasDaq : 9k€.
- Index_DAX : 8k€.
- Actions : 'buy' et 'sell' pour chaque combinaison de chaque indice. 'go_to_2ptc' pour l'action de vente totale de tous les indices.
- Rewards : toutes les récompenses sont nulles excepté pour l'action menant à un état final (pas de multi-objectif) parmi les 12 états identifiés (récompense à +1). L'état final est choisi tel que la somme des valeurs des indices est la plus importante (formule 1).
- learning rate $lr = 0.8$.
- $\gamma = 0.95$.
- 2000 itérations.
- état initial : 2 actions DAX et aucune CAC40 et NasDaq. choisi de représenter cet état par la liste [0,0,2].
- cash : 6k€, la valeur d'argent disponible non investi en action.

Les états possibles (auxquels il faut rajouter l'état 2%) sont calculés en considérant toutes les combinaisons possibles des indices dont la somme ne dépasse pas la CAF. Ils sont représentés dans le tableau 1. Par exemple, l'état 3 correspond à aucun indice CAC40 et DAX et un indice NasDaq.

L'état final calculé est celui qui permet d'obtenir la totalité du placement du cash (6k€) : [2,0,1]. Cet état correspond à un placement d'une valeur de 20k€en considérant la valeur des indices considérés.

Etat	CAC40	NasDaq	DAX
1	0	0	1
2	0	0	2
3	0	1	0
4	0	1	1
5	0	2	0
6	1	0	0
7	1	0	1
8	1	1	0
9	2	0	0
10	2	0	1
11	3	0	0

Tableau 1 – États possibles après analyse du scénario de simulation.

Action	CAC40	NasDaq	DAX
1	0	0	+1
2	0	+1	0
3	+1	0	0
4	+2	0	0
5	0	0	-1
6	0	-1	0
7	-1	0	0
8	-2	0	0

Tableau 2 – Actions possibles après analyse des états possibles.

Les actions possibles (auxquelles il faut rajouter les états go_to_2% et wait) sont calculés en considérant toutes les transitions possibles entre les états. Elles sont représentées dans le tableau 2. Lorsque le chiffre attribué à un indice est positif (resp. négatif), cela correspond à un achat (resp. vente). Par exemple, l'action 8 correspond à la vente de deux actions CAC40. Attention, la définition de ces actions ne prend pas en compte l'évolution de la variable cash qui évolue au gré des achats et des ventes (évolution dans le graphe d'états en fonction des actions). C'est la raison pour laquelle, la variable cash est introduite dans l'algorithme du Q-Learning présenté dans le listing 1 (lignes 16 à 23).

Le tableau 3 présente une politique possible après simulation. En effet, il existe d'autres politiques possibles après exécution de l'algorithme Q-Learning.

État	CAC40	NasDaq	DAX
[0,0,1]	buy 2	-	-
[0,0,2]	-	-	sell 1
[0,1,0]	go_to_2%	go_to_2%	go_to_2%
[0,1,1]	go_to_2%	go_to_2%	go_to_2%
[0,2,0]	go_to_2%	go_to_2%	go_to_2%
[1,0,0]	wait	wait	wait
[1,0,1]	buy 1	-	-
[1,1,0]	go_to_2%	go_to_2%	go_to_2%
[2,0,0]	go_to_2%	go_to_2%	go_to_2%
[2,0,1]	wait	wait	wait
[3,0,0]	sell 1	-	-
[3,0,1]	wait	wait	wait

Tableau 3 – Politique possible après simulation.

Comme l'état initial est [0,0,2], le tableau 3 nous propose de vendre une action DAX. Cette action nous donne une nouvelle valeur de la variable cash=6k€+8k€=14k€et un nouvel état qui est [0,0,1]. Toujours d'après le tableau 3, ce nouvel état impose d'acheter une action CAC40 (1 ligne du tableau 3) et conduit au nouvel état [1,0,1] avec un nouveau cash=14k€-6k€=8k€. Ensuite, cet état conduit à acheter une autre action CAC40 et donc d'atteindre l'état final [2,0,1] avec un nouveau cash=8k€-6k€=2k€. La suite d'action à réaliser est donc : vendre une action DAX, acheter une action CAC40 puis en racheter une autre. L'effet levier constitué réside dans le bénéfice réalisé par rapport au placement du cash initial dans un panier d'actions.

6 Conclusions et perspectives

Ce papier présente une modélisation d'un processus d'optimisation des actifs financiers afin de définir la politique optimale à appliquer pour effectuer un effet de levier sur les capacités d'autofinancement des acteurs économiques. Cette modélisation a été réalisée à l'aide d'un processus décisionnel Markovien modélisé grâce à DEVS et résolu via simulation basé sur de l'apprentissage par renforcement à l'aide de l'algorithme Q-Learning qui intègre la notion de capacité financière (cash) au cours du temps.

La poursuite de nos travaux devra permettre d'intégrer d'autres facteurs constitutifs de l'environnement qui influencent directement sur la construction d'une politique d'optimisation de la CAF comme par exemple les coûts de transaction, et le comportement des autres agents.

Dans ce sens l'intérêt majeur de l'utilisation de DEVS réside dans la possibilité pour construire un système composé d'un modèle couplé spécifiant l'environnement en connexion avec un modèle atomique représentant le comportement de l'agent. De plus, DEVS devrait nous offrir la possibilité de simuler la fréquence des prises de décision pour consentir à notre agent d'être plus rapide qu'un opérateur humain, qui prend ses décisions en se basant sur son intuition et sur des balises et plus intelligent que les algorithmes du trading à haute fréquence.

En perspective, les prochaines étapes de notre travail consisteront à ajouter au Q-Learning l'estimation de l'équation de Bellman par un réseau de neurones (Deep Q-Networks). En effet, l'algorithme du Q-Learning basé sur la gestion d'une table atteint ces limites lorsque le nombre d'état augmente. L'utilisation des réseaux de neurones permet d'avoir un approximateur de la variable Q et permet de considérer un nombre d'états possibles beaucoup plus important. Cette évolution devrait nous permettra non seulement de ne pas avoir à estimer en amont les récompenses mais aussi de répondre à l'accroissement du nombre d'états générés par la prise en compte de nouveaux facteurs constitutifs de l'environnement.

Références

[1] S. Jean, N. Mulder, and M. P. Ramos, "A general equilibrium, ex-post evaluation of the eu–chile free trade agreement," *Economic Modelling*, vol. 41, no. Supplement C, pp. 33 – 45, 2014.

[2] M. C. Fu, *Handbook of Simulation Optimization*. Springer Publishing Company, Incorporated, 2014.

[3] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation, Second Edition*. Academic Press, 2000.

[4] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *CoRR*, vol. abs/1604.00289, 2016.

[5] E. Rachelson, G. Quesnel, F. Garcia, and P. Fabiani, "A simulation-based approach for solving generalized semi-markov decision processes," in *Proceedings of the 2008 Conference on ECAI 2008 : 18th European Conference on Artificial Intelligence*, (Amsterdam, The Netherlands, The Netherlands), pp. 583–587, IOS Press, 2008.

[6] S. Meraji and C. Tropper, "A machine learning approach for optimizing parallel logic simulation," in *2010 39th International Conference on Parallel Processing*, pp. 545–554, Sept 2010.

[7] S. Toma, *Detection methodology and identify multiple faults in complex systems from discrete events and neural networks : applications for wind turbines*. Theses, Université Pascal Paoli, Sept. 2014.

[8] M. W. Floyd and G. A. Wainer, "Creation of devs models using imitation learning," in *Proceedings of the 2010 Summer Computer Simulation Conference*, SCSC '10, (San Diego, CA, USA), pp. 334–341, Society for Computer Simulation International, 2010.

[9] H. Saadawi, G. Wainer, and G. Pliego, "Devs execution acceleration with machine learning," in *2016 Symposium on Theory of Modeling and Simulation (TMS-DEVS)*, pp. 1–6, April 2016.

[10] B. P. Zeigler and H. S. Sarjoughian, "System entity structure basics," in *Guide to Modeling and Simulation of Systems of Systems*, Simulation Foundations, Methods and Applications, pp. 27–37, Springer London, 2013.

[11] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA : MIT Press, 1st ed., 1998.

[12] D. P. Bertsekas, *Dynamic Programming : Deterministic and Stochastic Models*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1987.

[13] M. L. Puterman, *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. New York, NY, USA : John Wiley & Sons, Inc., 1st ed., 1994.

[14] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA : Princeton University Press, 1 ed., 1957.

[15] L. Capocchi, J. F. Santucci, B. Poggi, and C. Nicolai, "DEVSimPy : A collaborative python software for modeling and simulation of DEVS systems," in *Proc. of 20th IEEE International Workshops on Enabling Technologies*, pp. 170–175, June 2011.