| | |
|---|---|
| **Titre:** Title: | Utilisation de l'apprentissage automatique pour la planification de projet de construction |
| **Auteur:** Author: | Joffrey Clément |
| **Date:** | 2021 |
| **Type:** | Mémoire ou thèse / Dissertation or Thesis |
| **Référence:** Citation: | Clément, J. (2021). Utilisation de l'apprentissage automatique pour la planification de projet de construction [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/9237/ |

**POLYTECHNIQUE MONTRÉAL**

affiliée à l'Université de Montréal

# Utilisation de l'apprentissage automatique pour la planification de projet de construction

**JOFFREY CLÉMENT**

Département de mathématiques et de génie industriel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

Génie industriel

Septembre 2021

**POLYTECHNIQUE MONTRÉAL**

affiliée à l'Université de Montréal

Ce mémoire intitulé :

# Utilisation de l'apprentissage automatique pour la planification de projet de construction

présenté par **Joffrey CLÉMENT**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

**Pierre BAPTISTE**, président
**Robert PELLERIN**, membre et directeur de recherche
**Bernard GRABOT**, membre et codirecteur de recherche
**Alain HAÏT**, membre

# REMERCIEMENTS

Je tiens à remercier toutes les personnes importantes qui m'ont soutenu et aidé tout au long de cette maîtrise qui m'a énormément appris, que ce soit sur la gestion de projet ou sur mes capacités à mener un projet de recherche.

Tout d'abord, je remercie mon directeur de recherche, Monsieur Robert Pellerin qui m'a offert la possibilité d'effectuer cette maîtrise dans les meilleures conditions qui soient et qui a su faire preuve de beaucoup de patience, surtout dans les derniers mois de ma maîtrise qui a connu beaucoup de rebondissements. Dès notre première rencontre en novembre 2017, bien avant que je commence une maîtrise, il a su me conseiller et me guider pour construire le projet de recherche que je présente aujourd'hui et qui est en totale adéquation avec les objectifs que je pouvais avoir avant de commencer ma maîtrise.

Je souhaite également remercier Nathalie Perrier ainsi que Bernard Grabot pour leur aide, leur soutien ainsi que leurs bons conseils tout au long de la maîtrise.

Je tiens aussi à remercier toute l'équipe du bureau : Marine Echternach, Mathieu Lalagüe et Lucas Guilhen. Merci à vous de m'avoir permis de travailler dans une ambiance très agréable et motivante quotidiennement, même alors que les conditions sanitaires nous ont empêchées de nous voir pendant longtemps.

Il est également important pour moi de remercier ma famille, en particulier ma mère qui m'a toujours soutenu tout au long de mes études malgré la distance.

À Polytechnique Montréal, j'ai également vécu une expérience intense au sein de l'Association des Étudiants des Cycles Supérieurs de Polytechnique Montréal dont je remercie toute l'équipe. Je remercie tout particulièrement Simona qui a tant fait de travail pour moi de façon que je puisse me concentrer sur ma maîtrise.

Enfin, je tiens à terminer ce chapitre en remerciant mes colocataires Romain et Simon ainsi que mon amie Léa qui m'ont connu dans tous les états de stress et de motivation, mais qui ont su rester derrière moi afin de m'encourager afin de terminer cette maîtrise après tant d'émotions.

# RÉSUMÉ

L'industrie de la construction ne se repose plus simplement sur un mode de construction pour lequel toutes les activités se déroulent sur le site de construction. Par exemple, la construction modulaire consiste à préparer des morceaux de l'édifice, appelés modules, en usine avant de les transporter sur le site. Ces modes de construction alternatifs commencent à être utilisés de plus en plus et on comprend que les besoins en termes de planification diffèrent de la construction plus classique.

Mais le mode de construction est choisi dès les prémices du projet et la décision se concentre sur l'expérience de l'entreprise et non sur des enjeux de planification qui pourrait montrer des avenues intéressantes. Ainsi, l'objectif de ce travail est de proposer un modèle permettant de générer des échéanciers dans le contexte de planification initiale en considérant les modes de construction alternatifs.

Afin de remplir cet objectif, on a commencé par définir un cadre de planification adapté à la problématique pour laquelle on a fait une analogie avec la théorie des jeux. Le modèle a ensuite été développé avant de l'expérimenter et le valider grâce à des instances de projet générées spécifiquement dans le cadre de ce travail. Enfin, on le compare à d'autres modèles afin d'en mesurer les performances.

Le modèle proposé s'appuie sur la combinaison d'un arbre de recherche de Monte-Carlo et d'un réseau de neurones dans un processus d'apprentissage automatique par renforcement. Ainsi, le modèle apprend et propose une planification de plus en plus performante. En effet, tout comme AlphaGo joue aux échecs en choisissant les pièces à jour, notre modèle identifie itérativement le lot de travail à planifier en s'appuyant sur l'arbre de recherche de Monte-Carlo qui permet de représenter l'ensemble des possibilités à partir d'un état de planification. L'arbre est aidé par le réseau de neurones en calculant les probabilités des lots de travaux ainsi que la valeur des états de planification suivants, afin de faciliter le choix du lot de travail à planifier et en évitant une recherche exhaustive.

Le modèle a montré qu'il peut s'améliorer et converger presque systématiquement après plusieurs itérations. Le nombre de simulations de l'arbre de recherche de Monte-Carlo ainsi que le nombre d'exemples à générer au début de chaque itération apparaissent comme des paramètres déterminants afin de garantir un bon apprentissage. Enfin, on a montré que le modèle est plus

performant qu'un choix aléatoire, mais connait des difficultés face au modèle choisissant systématiquement le mode de construction classique. L'ensemble de ce travail a été concentré sur la proposition d'un premier modèle, certains éléments comme la structure et la configuration du réseau de neurones n'ont pas été optimisés ce qui constitue des perspectives de recherche intéressantes pour la suite.

# ABSTRACT

The construction industry no longer relies simply on a method of construction in which all activities take place at the construction site. For example, modular construction involves preparing pieces of the building, called modules, in a factory before transporting them to the site. These alternative construction methods are beginning to be used more and more and it is understood that the planning requirements are different from more traditional construction mode.

But the construction method is chosen at the beginning of the project and the decision is based on the company's expertise and not on planning issues that could show interesting avenues. Thus, the objective of this work is to propose a model to generate schedules in the initial planning context by considering alternative construction modes.

In order to achieve this objective, we started by defining a planning framework adapted to the problem for which an analogy with game theory was made. The model was then developed before experimenting and validating it with project instances generated specifically for this work. Finally, we compare it to other models in order to measure its performance.

The model is based on the combination of a Monte-Carlo search tree and a neural network in an automatic learning process by reinforcement. Thus, the model learns and generates an increasingly efficient planning. Indeed, just as AlphaGo plays chess by choosing the pieces to be updated, our model iteratively identifies the batch of work to be planned by relying on the Monte-Carlo search tree which allows representing the set of possibilities from a planning state. The tree is helped by the neural network by calculating the probabilities of the work packages as well as the value of the following planning states, in order to facilitate the choice of the work package to be planned and by avoiding an exhaustive search.

The model has shown that it can improve and converge almost systematically after several iterations. The number of simulations of the Monte-Carlo search tree as well as the number of examples to be generated at the beginning of each iteration appear to be determining parameters in order to guarantee a good learning. Finally, we have shown that the model performs better than a random choice, but has difficulties in comparison with the model that systematically chooses the classical construction mode. The whole of this work was concentrated on the proposal of a first model, some elements like the structure and the configuration of the neural network were not optimized which constitutes interesting research opportunities for the continuation.

# TABLE DES MATIÈRES

# LISTE DES TABLEAUX

# LISTE DES FIGURES

# LISTE DES SIGLES ET ABRÉVIATIONS

| | |
|---|---|
| CPM | Critical path method |
| FEL | Front End Loading |
| IPA | Independent Project Analysis |
| MBI | Modular Building Institute |
| MCTS | Monte Carlo Tree Search |
| MRCPSP | Multi mode Resource Constraint Planning Scheduling Problem |
| RCCP | Rough-Cut Capacity Planning |
| RCPSP | Resource Constraint Project Scheduling Problem |
| WBS | Work Breakdown Structure |

# LISTE DES ANNEXES

# CHAPITRE 1 INTRODUCTION

L'industrie de la construction occupe une place importante dans notre société. Nous avons sans cesse besoin de construire de nouveaux logements pour la population grandissante ainsi que des infrastructures pour accompagner le développement des entreprises. Ce secteur est donc logiquement l'un des plus gros secteurs économiques, représentant plus de 12 % du PIB au Québec (Ccq, 2017). Mais ce secteur connait de nombreuses difficultés, qui sont principalement le manque de main-d'œuvre, une productivité stagnante, des problèmes de sécurité ainsi qu'une adoption des nouvelles technologies trop lente (Turner, C. J. et al., 2021).

En général, les grands projets de construction se décomposent en trois phases : la planification initiale, puis l'exécution avec les phases d'ingénierie, d'approvisionnement et de construction et enfin la mise en service. Dans la phase de planification initiale et le début de la phase d'exécution, les grands projets de construction sont caractérisés par un haut niveau d'incertitude, étant donné que les choix de conception et de modes d'exécution ne sont pas tous établis. Pour faire face à ces défis, les gestionnaires des grands projets ont généralement recours à un processus de planification hiérarchique. Ainsi, des échéanciers agrégés sont définis à des horizons longs terme, alors que d'autres échéanciers plus détaillés sont développés pour le court terme au fur et à mesure de l'avancement du projet.

Parmi les décisions de conception, le recours ou non à la préfabrication doit être envisagé rapidement. Cette technique consiste à fabriquer des parties de la construction en usine, puis à les assembler sur le site. Elle présente de nombreux avantages, car elle augmente la sécurité sur les chantiers (Bikitsha, L. et al., 2011), permet un gain de qualité, une baisse de la durée des projets et parfois même une baisse des coûts (Hwang, B. G. et al., 2018). Elle apparaît alors comme une solution idéale aux défis du secteur de la construction, mais plusieurs obstacles s'opposent à son adoption. Une particularité de la préfabrication est qu'elle remet en question les processus de construction classiques. Elle a des implications très importantes sur la planification de projet, car les activités réalisées sur le site de construction et hors site ne sont plus les mêmes ; les parties prenantes peuvent également être différentes, tout comme les ressources nécessaires.

Son utilisation doit alors être décidée très tôt lors de la phase de planification initiale, avant même que l'ingénierie détaillée ne soit terminée. Mais le choix du mode de construction est souvent fait en fonction de l'expertise de l'entreprise et des caractéristiques globales du projet, sans forcément

comparer toutes les alternatives possibles du point de vue de la planification (Mikulakova, Eva et al., 2010). De plus, dans le cadre des grands projets de construction, un élément de complexité se manifeste par un très grand nombre d'échéanciers différents, mais menant tous à l'achèvement du projet. Ce nombre de possibilités augmente alors naturellement en prenant en considération la préfabrication, ce qui complexifie encore l'élaboration des échéanciers.

Toutes ces possibilités s'accompagnent également d'opportunités pour définir des échéanciers encore plus réalistes et performants.

Dans ce contexte, ce projet de recherche a pour ***objectif de montrer la faisabilité d'un outil basé sur l'apprentissage automatique permettant au gestionnaire de projet de planifier un projet de construction en prenant en compte les différents modes de construction tels que la préfabrication***.

La suite du mémoire est organisée comme suit. Premièrement, le Chapitre 2 présente une revue de la littérature du domaine. Ce chapitre débute avec la présentation des concepts de planification de projets applicables au secteur de la construction. Puis, les différentes approches proposées pour faciliter la prise de décision concernant les modes de construction sont présentées et analysées afin d'en faire ressortir les limitations pour notre contexte d'étude. Le Chapitre 3 présente les objectifs de recherche ainsi que la méthode adoptée pour ce projet de recherche permettant de valider l'intérêt du modèle développé pour aider à la planification de grands projets de construction en considérant différents modes de construction. Le modèle développé dans ce projet de recherche est présenté au Chapitre 4. Dans le Chapitre 5, le modèle proposé est testé afin de juger de son intérêt. Cette expérimentation permet de comparer les résultats du modèle à ceux d'autres modèles très simples comme des planifications aléatoires. Finalement, le Chapitre 6 fait une synthèse des travaux réalisés, souligne les limites de l'étude et introduit des perspectives de recherche.

# CHAPITRE 2 REVUE DE LA LITTÉRATURE

Ce chapitre présente une revue des différentes approches proposées pour la planification de projet de construction en considérant les modes de construction alternatifs. Après avoir décrit le contexte ainsi que les concepts clés de ce mémoire, la revue se divise en quatre parties présentant dans l'ordre les approches de planification tactique, les approches d'ordonnancement adaptées à la construction modulaire, les approches d'aide à la décision et enfin les approches de valorisation des données de planification. Le chapitre se conclut avec une analyse critique qui permet de mettre en avant les limitations de ces approches et ainsi identifier des opportunités de recherche.

## 2.1 Contexte

### 2.1.1 Planification des grands projets de construction

Pour faciliter le processus de planification des grands projets de construction et d'ingénierie, la planification hiérarchique est souvent utilisée afin d'élaborer différents échéanciers au cours du projet. Un échéancier rassemble les dates de début et de fin des phases, activités ou lots de travaux de l'ensemble du projet. Suivant le type d'échéancier, il peut être plus ou moins agrégé de manière à représenter un niveau de détail plus ou moins important.

Dans le cadre de la planification hiérarchique, des échéanciers agrégés sont alors développés pour le long terme alors que d'autres sont développés pour le court terme. Les échéanciers sont de plus en plus détaillés à mesure que l'horizon de planification se rétrécit en passant de l'ensemble du projet aux étapes et phases de projet (Leon, Gui Ponce De, 2011).

Des modèles permettant la génération d'échéanciers agrégés au niveau tactique existent dans la littérature. Ils sont généralement associés au problème de *Rough-Cut Capacity Planning* (RCCP). Dans ce problème, les activités détaillées ne sont pas considérées directement, mais elles sont incluses dans des groupes d'activités appelés lots de travaux. De même, les capacités de ressources ne sont pas connues de manière exacte, elles sont simplement estimées sur les périodes agrégées.

Ici, le niveau tactique correspond aux phases d'appel d'offres et d'acceptation de projet (De Boer, Ronald, 1998) qui sont incluses dans la phase de pré-exécution. Dans le cas de grands projets de construction, qui est le point d'intérêt de ce mémoire, le niveau de planification tactique est élargi

et concerne également les premières phases d'exécution pendant lesquelles les travaux d'ingénierie ne sont habituellement pas encore terminés.

La planification au niveau tactique vise ainsi à obtenir les dates de jalons et à valider les budgets du projet avec l'ensemble des parties prenantes en cherchant à planifier les lots de travaux et non les activités. Ce niveau de planification est bien différent du niveau opérationnel dont l'objectif est de développer des plans de travail dans un horizon plus court. Pour ce dernier niveau, des techniques d'ordonnancement sont développées dans la littérature et correspondent notamment au *Resource Constraint Project Scheduling Problem* (RCPSP). Ce problème a été largement traité dans la littérature (Pellerin, Robert et al., 2020), mais ne correspond pas au contexte de cette présente étude.

Les grands projets de construction suivent généralement trois phases de gestion de projet : pré-exécution, exécution et mise en service. La phase de pré-exécution connait de nombreuses appellations selon les industries. Le terme le plus utilisé dans l'industrie est *Front End Loading* (FEL). *Independent Project Analysis* (IPA), un cabinet de conseil reconnu en gestion de projet, définit le FEL comme étant le processus permettant de définir précisément un projet de façon à atteindre les objectifs d'affaires de la compagnie. C'est l'occasion de répondre aux questions pourquoi, quoi, quand, comment, où et qui d'un projet (Van Der Weijde, Ga, 2008).

Le standard FEL est généralement divisé en 4 grandes phases : FEL0, FEL1, FEL2, FEL3. Chaque fin de phase correspond à une décision menant à la poursuite ou à l'arrêt du projet. Les phases FEL0 et FEL1 permettent de valider l'importance stratégique du projet ainsi que sa viabilité. L'objectif de la phase FEL2 est de développer le projet en évaluant les différentes alternatives de conception, de commercialisation et de processus de construction. La meilleure alternative est alors sélectionnée pour la phase FEL3 et sera détaillée afin d'obtenir la validation et l'investissement du client. C'est dans la phase de FEL3 que les premiers plans d'exécution sont finalisés (De Boer, Ronald, 1998; Gibson Jr, Ge et al., 1995; Spangler, Ryan, 2005).

La succession de ces phases permet donc de développer différents niveaux d'échéanciers (Aace International, 2010). Chaque classe d'échéancier est définie par rapport au pourcentage approximatif de l'avancement de la conception du produit et correspond aux phases du FEL.

Ainsi, l'échéancier de classe 5 est développé durant le FEL0 et est basé sur l'information minimum. L'objectif, l'étendue et les risques du projet sont définis succinctement de façon à pouvoir démarrer

le projet. La visée est ici principalement stratégique et l'échéancier est représenté sous la forme d'une succession de grandes phases, ou sous forme d'un diagramme de Gantt sans contraintes de précédence. Seuls les grands jalons du projet sont utilisés et cet échéancier est défini selon les directives de la direction de l'entreprise en suivant une approche descendante de planification.

L'échéancier de classe 4 est développé dans le but de faciliter l'étude de faisabilité. Il est bien plus détaillé et se base sur une structure formelle de découpage de projet, appelée couramment *Work Breakdown Structure* (WBS). Ce WBS présente une structure hiérarchisée de l'ensemble des lots de travaux à réaliser lors du projet. Il est également développé à partir d'une planification descendante et fait apparaitre les livrables de haut niveau. Les échéanciers de classe 4 sont généralement développés lors des phases FEL1 et FEL2.

L'échéancier de classe 3 est en général le dernier échéancier avant la phase d'exécution. Il correspond donc à la phase du FEL3. Il est un peu plus détaillé que l'échéancier de classe 4. Il sert en général à appuyer les demandes de financement et à contrôler le déroulement du projet d'un point de vue global. Les méthodes utilisées pour les échéanciers de classe 1, 2 et 3 comprennent également la méthode de chemin critique (CPM) et la méthode PERT. CPM permet d'estimer la durée minimum du projet en identifiant la longueur de la plus longue séquence appelée chemin critique. Les durées des activités sont déterministes et les ressources sont supposées illimitées. La méthode PERT est basée sur la méthode CPM en considérant en plus des durées probabilistes avec trois possibilités : une valeur optimiste, une valeur pessimiste et une valeur la plus probable. L'objectif de cette méthode est de mesurer la probabilité que le projet se finisse dans des délais fixés (Cottrell, Wayne D, 1999).

Les deux dernières classes d'échéancier font appel à des approches ascendantes pour la planification qui se définissent par la participation active de l'équipe opérationnelle dont les besoins de planification sont remontés afin de déterminer les échéanciers.

Ces échéanciers sont alors détaillés et servent à contrôler le bon déroulement de la construction, mais sont également utilisés pour les appels d'offres. Les échéanciers de classe 2 sont utilisés au début de la construction alors que les échéanciers de classe 1 servent à coordonner les opérations quotidiennes. Ces derniers sont les échéanciers les plus précis et nécessitent que le produit soit entièrement défini ou presque.

On peut retrouver les définitions synthétiques des classes d'échéancier dans le Tableau 2.1.

Dans le contexte de grands projets de construction et d'ingénierie, la complexité est telle qu'une structure avec de multiples échéanciers diversifiés est nécessaire pour communiquer de manière appropriée avec l'ensemble des parties prenantes au projet durant les différentes phases.

Tableau 2.1 Définition des classes d'échéanciers, adaptée de (Aace International, 2010)

| Classe d'échéancier | Degré de définition du projet (en % de la définition complète) | Utilisation finale | Méthode de planification utilisée |
|---|---|---|---|
| Classe 5 | 0% à 2% | Sélection de concept | Planification descendante |
| Classe 4 | 1% à 15% | Étude de faisabilité | Planification descendante Semi détaillée |
| Classe 3 | 10% à 40% | Budget, autorisation ou contrôle | Planification descendante Semi détaillée |
| Classe 2 | 30% à 70% | Contrôle ou soumission/appel d'offres | Planification ascendante Détaillée |
| Classe 1 | 50% à 100% | Contrôle ou soumission/appel d'offres | Planification ascendante Détaillée |

### 2.1.2 Construction alternative et classique

Traditionnellement, les activités de construction d'un bâtiment se déroulent très majoritairement sur le site de construction. Le début consiste à préparer le site de construction en mettant en place les mesures de sécurité et en démolissant les éléments pouvant gêner la construction. Ensuite, les fondations sont faites avec du ciment directement coulé sur place. L'étape suivante consiste à installer la structure du bâtiment en plaçant les poutres avant de construire les murs porteurs. Une

fois l'ensemble des murs construit, il est possible d'installer le toit. Enfin, l'aménagement intérieur est réalisé. En parallèle de chaque étape, toutes les infrastructures sont installées pour permettre l'accès à l'eau et à l'électricité dans le bâtiment. Le bâtiment est terminé et opérationnel après ces étapes, toutes réalisées sur le site de construction. Bien évidemment, en fonction des particularités du bâtiment, les étapes peuvent différer légèrement, mais il s'agit là du schéma classique utilisé depuis bien longtemps. Ce mode de construction possède bien des avantages, notamment car il est parfaitement maitrisé. De plus, l'accessibilité de toutes les ressources sur le site permet de simplifier la gestion pour le chef de chantier.

Mais il existe également des inconvénients intrinsèques à ce mode de construction. Par exemple, le bon déroulement de la construction et très dépendant de la météo, notamment au début de la construction, avant que le toit soit construit. La météo est une variable indépendante du gestionnaire de projet, mais elle peut avoir de lourdes conséquences sur la durée et le coût du projet. De plus, certaines activités demandent du temps de séchage pendant lequel aucune autre ne peut être effectuée sur l'espace occupé. Les tâches effectuées sur le site de construction génèrent souvent de la poussière qui limite grandement la possibilité d'obtenir un degré de précision parfois nécessaire pour l'installation ou la découpe de certains éléments. Il existe également des enjeux de sécurité. En effet, de nombreux accidents surviennent sur les chantiers de construction, notamment lors de la construction d'édifice à plusieurs étages, malgré des consignes de sécurité strictes (Winge, Stig et al., 2019). Enfin, l'utilisation du mode de construction classique amène également des problèmes de productivité. En effet, certains rapports démontrent que très peu d'activités à valeur ajoutée sont effectuées sur le site de construction (Cefrio, 2011). Ainsi, il peut être intéressant de remettre en question le choix d'effectuer la majorité des activités sur le chantier.

Dans ce sens, il existe des modes de construction alternatifs, comme la construction modulaire. Cette approche consiste en la fabrication de macro-éléments en atelier, avant de les transporter et de les monter sur le site de construction. Cette approche tient ses origines au début du XXe siècle avec la première maison Dom-Ino imaginée par Le Corbusier en 1914, qui s'appuie sur la combinaison d'éléments aux dimensions standardisées (Gans, D. et al., 2006). Par la suite, cette dynamique a émergé avec la construction de 12 000 logements dans la Nouvelle Francfort entre 1925 et 1933, ainsi qu'avec la maison construite avec le système Förster-Kraft en 1932. Cette maison était basée sur des panneaux modulaires en bois de la taille d'un étage. De plus en plus d'initiatives ont participé à élaborer ces nouveaux modes de construction. À partir d'un ensemble

de projets européens, Gosling a proposé une définition pour ces nouveaux modes de construction qui rassemble la préfabrication, le préassemblage, la modularisation et la fabrication hors site.

Ces modes de construction possèdent de nombreux avantages. Le *Modular Building Institute (MBI)* a listé les principaux dans son rapport de 2020 (Modular Building Institute, 2020). Ce rapport a été constitué sur la base d'un sondage auprès de tous les types d'acteurs de la construction, pour leur demander l'impact de la construction modulaire et de la préfabrication. Sans ambiguïté, la majorité des acteurs s'accorde à dire que la construction modulaire et la préfabrication ont un fort impact positif sur la réduction des déchets, la productivité, la qualité, la satisfaction du client, l'évaluation des coûts et la planification, ainsi que sur la sécurité.

Ces avantages ont été confirmés par d'autres articles dans la littérature scientifique. Notamment, Choi, J. O. et al. (2019) mettent en avant une diminution de la durée de projet et du coût à condition d'une bonne gestion de projet. Gibb, Alistair Gf (1999) a aussi mis en avant les avantages de la préfabrication et de la construction hors site comparativement à la construction classique sur le chantier. Il liste des avantages en termes de temps, coût, qualité, sécurité, productivité dans le cadre de projets réplicables.

## 2.2 Revue des approches de planification de projet de construction considérant les modes de construction alternatifs.

### 2.2.1 Classification des approches identifiées

Les approches de planification identifiées dans le cadre de cette revue peuvent se répartir selon plusieurs catégories. Tout d'abord, il existe des variantes des approches de planification hiérarchique qui considèrent des modes d'exécution multiples.

Par ailleurs, la littérature scientifique contient de nombreux travaux répondant au problème d'ordonnancement classique qu'est le RCPSP. À partir d'un ensemble d'activités liées entre elles par des relations de précédence ainsi que de ressources limitées, l'objectif de ces approches est de déterminer les dates de début des activités afin de minimiser la durée totale du plan de projet (De Boer, Ronald, 1998). Bien que ce problème se concentre au niveau des activités, certaines variantes considèrent des modes multiples pour les activités qui se rapprochent des modes de construction

alternatifs considérés dans ce projet de recherche. Ce rapprochement mérite d'être étudié pour évaluer la capacité de ces approches à planifier des projets en considérant les différents modes de construction.

La catégorie d'approche suivante est l'identification des facteurs influençant la planification de projet en fonction du mode de construction. Ces facteurs sont des caractéristiques du projet ainsi que des éléments du contexte dans lequel le projet se déroule.

Enfin, une des caractéristiques du problème qui est relevé ici concerne le grand nombre de possibilités envisageables pour les échéanciers de projet, qui dépendent bien évidemment du nombre de modes de construction considérés, mais aussi de la taille du projet. Dans le cadre de grands projets de construction, quand on envisage de choisir un mode de construction pour chaque lot de travail, le nombre de possibilités est naturellement gigantesque. Il est alors indispensable de pouvoir traiter un grand nombre de données, que ce soit pour réutiliser les données de projet historiques ou pour évaluer toutes les possibilités. Des approches basées sur l'apprentissage automatique, *machine learning*, ont été testées dans ce sens. Certaines utilisent les données de projets passés pour prédire la durée du projet en fonction de ses caractéristiques, dont le mode de construction. Parmi elles, certaines approches utilisent les réseaux de neurones pour optimiser la recherche du meilleur plan de projet. Ces réseaux de neurones utilisent les données pour proposer des échéanciers de projets très performants sans avoir à évaluer toutes les possibilités.

### 2.2.2 Les approches de planification hiérarchique

Dans le contexte des grands projets de construction, les approches de planification hiérarchique consistent à développer des échéanciers agrégés suivant les différentes phases du projet, avec un niveau de précision variable. Ces méthodes s'attaquent en général au problème RCCP et certains auteurs ont proposé des approches pouvant s'adapter au contexte de modes multiples pour la construction.

Une de ces approches consiste à proposer un modèle de planification tactique adapté au contexte complexe des grands projets d'ingénierie et de construction en intégrant plusieurs niveaux d'agrégation des données à travers la variation des durées des périodes considérées, très présentes au niveau tactique, lorsque des décisions sont prises alors que les connaissances sur le projet sont

encore limitées. Des variantes de ce modèle ont également été développées pour faire face aux incertitudes de charges de travail (Cherkaoui, Kaouthar, 2017). Les résultats de cette étude ont montré l'intérêt d'utiliser des échéanciers agrégés pour les grands projets de construction, incluant une nette amélioration des temps de calcul (Cherkaoui, Kaouthar, 2017).

On peut également noter que d'autres approches ont été proposées pour la planification à différents niveaux hiérarchiques. En effet, Sunke, Nicole (2008) propose de résoudre le problème d'ordonnancement de projet au niveau tactique sans considérer les contraintes de capacités, en utilisant le CPM ou PERT. Également, Speranza, M Grazia et al. (1993) proposent un modèle au niveau tactique pour le problème d'ordonnancement dans le contexte multi-projets. Ils supposent que les projets sont multimodes. Chaque projet correspond alors à une activité avec plusieurs modes possibles. L'objectif est donc de déterminer les dates de début et les modes de chacun des projets afin de maximiser la valeur actuelle nette totale.

### 2.2.3 L'ordonnancement adapté à la construction modulaire

#### 2.2.3.1 **Le multi mode RCPSP**

Le problème RCPSP est un problème d'ordonnancement connu qui a pour but de déterminer les dates de début d'activités de façon à minimiser la durée totale d'un projet. Certains auteurs ont proposé des approches intéressantes adaptées à la construction modulaire, à travers une variante du RCPSP qui est le RCPSP multi mode (*Multimode Resource Constraint Planning Scheduling Problem* - MRCPSP).

Cette variante a été initialement définie par Elmaghraby, Salah E (1964). Elle considère des relations de précédence fixes, mais pour chaque activité, plusieurs modes d'exécution sont possibles. Un coût ainsi qu'une durée sont associés à chacun de ces modes d'exécution, qui pourraient être assimilés à des modes de construction.

Pour résoudre ce problème, beaucoup d'auteurs ont proposé des méthodes différentes qui ont été répertoriées par Pellerin, Robert et al. (2020). Le RCPSP étant un problème de complexité NP, les approches pour le résoudre sont basées sur des heuristiques ou des métaheuristiques, ces dernières s'avérant souvent particulièrement performantes.

Par exemple, Alcaraz, Javier et al. (2003) ont proposé une approche de résolution basée sur un algorithme génétique. D'autres comme Bouleimen, Klein et al. (2003) ont utilisé le recuit simulé. Comme pour le problème RCPSP, beaucoup d'articles présentent des approches de résolution différentes avec des performances nuancées suivant les caractéristiques des projets.

#### 2.2.3.2 **Le séquencement des lots de travaux**

Une autre variation du RCPSP intéressante est le *parallel machine scheduling problem*. Ce problème bien connu dans l'industrie manufacturière consiste à gérer l'utilisation de plusieurs machines en parallèle de façon à atteindre les objectifs de production. Hammad, A. W. A. et al. (2017) ont proposé une résolution de ce problème adaptée à la construction modulaire. Pour eux, la production en usine a beaucoup de similitudes avec la production des modules de construction. Ainsi, leur méthode permet de planifier la construction des modules de façon à les livrer à un moment choisi sur le chantier.

### 2.2.4 Les approches d'identification de facteurs différenciants

Pour aider le gestionnaire de projet concernant le choix du mode de construction, certains auteurs ont travaillé à l'identification des facteurs influençant le succès ou l'échec d'un projet en préfabrication.

Pour cela, Song, Jongchul et al. (2005) ont classé ces facteurs en dix catégories : la planification, le coût, la main-d'œuvre, la sécurité, les attributs du site, le système mécanique, le type du contrat de projet, la conception, les besoins de transports et la capacité des fournisseurs. À partir de ces catégories, ils ont développé un outil basé sur un questionnaire permettant au gestionnaire de projet de connaitre les facteurs qui seront déterminants dans son projet s'il choisit la préfabrication.

Murtaza, Mirza B et al. (1993) ont fait un travail similaire particulièrement adapté aux constructions dans l'industrie pétrochimique. Ils ont également identifié certains facteurs permettant au gestionnaire d'évaluer la faisabilité technique de la modularisation pour le projet, ainsi que les impacts relatifs sur le coût et la durée du projet. Ces facteurs sont listés sous cinq catégories reliées au lieu de la construction, à la main-d'œuvre, aux caractéristiques du projet, aux risques du projet ainsi qu'à la structure organisationnelle du projet.

D’autres travaux ont permis de mettre en évidence des outils d’aide à la décision concernant l’utilisation de la modularisation. Li, L. et al. (2018) ont identifié des facteurs critiques très différents, principalement basés les compétences et l’expérience des gestionnaires de projet. Ces facteurs sont l’expérience des concepteurs, l’expérience du fabricant, les compétences du gestionnaire de projet, la maturité des technologies utilisées ainsi que les politiques d’incitations.

Ces approches permettent d’identifier les facteurs utilisés afin de choisir le mode de construction. On comprend alors que les enjeux de planification ne sont pas pris en compte pour ce choix.

### 2.2.5 Approches de valorisation des données

#### 2.2.5.1 **Application de l’apprentissage automatique pour la planification**

L’apprentissage automatique pour la planification de projet dans la construction est appliqué dans un cas réel par Smith (2018). Partant d’une liste de tuyaux à installer et considérant l’effort nécessaire pour installer chaque élément, l’algorithme proposé a pour objectif de générer la séquence la plus courte. Des résultats très intéressants sont obtenus, car les séquences les plus performantes sont très différentes des séquences classiques. En effet, les meilleures séquences installent en priorité les éléments au centre avant de s’éloigner petit à petit du centre suivant un cercle, contrairement aux séquences connues dans la réalité qui commencent par installer les éléments proches du sol avant de monter petit à petit vers les éléments les plus hauts.

Cette heuristique combine un arbre de recherche de Monte-Carlo et un réseau de neurones. Chaque nœud de l’arbre correspond à un état de la planification pour lequel les pièces du bâtiment précédemment planifiées et celles restant à planifier sont connues et accompagnées de leur contrainte d’installation calculée dans le modèle. Ainsi, il est possible de mesurer la performance de la planification pour chaque nœud.

À partir d’un nœud, pour choisir la prochaine pièce à installer, l’algorithme parcourt et développe l’arbre de recherche lors de multiples itérations à la suite desquelles le choix de la pièce à installer est fait de façon à maximiser les chances que l’installation de cette pièce participe à la meilleure séquence, c’est-à-dire la plus rapide.

À chaque itération lors du choix d’une pièce, l’algorithme parcourt l’arbre jusqu’à une feuille en choisissant les nœuds suivant une politique d’évaluation des nœuds qui combine l’exploration de

nouveaux nœuds menant à de nouvelles séquences et l'exploitation des nœuds déjà visités pour obtenir une meilleure évaluation des séquences associées à ces nœuds. Il s'agit de l'étape de sélection. Une fois la feuille atteinte, l'algorithme complète l'arbre avec les nœuds possibles à partir de cette feuille qui correspond au choix des prochaines pièces à installer. Il s'agit de la phase d'expansion. L'étape suivante est la simulation, au cours de laquelle l'algorithme attribue à la feuille sa probabilité de faire partie d'une bonne séquence. Cette valeur est calculée par le réseau de neurones en fonction de l'état de planification au niveau de cette feuille. Pour ce faire, le réseau de neurones reçoit en entrée l'état de planification, c'est-à-dire l'ensemble des pièces déjà installées ainsi que celles restantes, avec leurs caractéristiques. Ce réseau de neurones est constitué d'un ensemble de couches de convolution ainsi que des couches résiduelles. En sortie, le réseau de neurones fournit un vecteur de probabilités indiquant la probabilité de chacune des pièces restantes d'être planifié à partir de cet état. Le réseau de neurones fournit également une valeur de l'état de planification.

À partir de cette valeur, l'algorithme met à jour l'ensemble des nœuds de l'arbre qui ont été visités lors de cette itération, ce qui modifie les poids des nœuds lors de leur choix suivant la politique d'évaluation des nœuds de l'étape de sélection. Cette dernière étape est la rétropropagation. Ces quatre étapes (sélection, expansion, simulation et rétropropagation) sont répétées à chaque itération. Dans l'exemple d'application de Smith (2018), le réseau de neurones est alors implémenté pour les étapes de sélection et l'étape de simulation.

Ce modèle d'algorithme est très proche de celui d'AlphaGo (Silver, David et al., 2017; Smith, 2018). Ce modèle, développé par DeepBlue, a permis de développer une intelligence artificielle capable de battre les meilleurs joueurs de Go. Pour appliquer ce modèle à la planification, Smith (2018) fait le rapprochement entre le choix d'une pièce à installer et le choix d'un coup de Go à jouer.

#### 2.2.5.2 **Prédiction de durée et de coût**

L'intelligence artificielle a aussi permis d'effectuer des prédictions sur la durée et le coût d'un projet en fonction de ses caractéristiques. Les articles retenus ont la particularité d'avoir utilisé des algorithmes d'apprentissage automatique à partir de données réelles.

Guo, Jian-Xia et al. (2019) ont utilisé 90 projets de construction dans toute la Chine pour entraîner leur réseau de neurones à prédire la durée d'un projet à partir du coût, du nombre d'étages, de la hauteur, de la surface du bâtiment, de la région, ainsi que du type de la structure.

Le Tableau 2.2 recense les auteurs ayant fait des travaux similaires, en se basant tous sur des données réelles. Les auteurs soulèvent le fait que malgré les progrès réalisés avec l'avènement de la Construction 4.0, il reste difficile de récolter des données réelles fiables dans la construction notamment à cause d'une importante résistance au changement dans cette industrie, mais aussi de la grande hétérogénéité du développement digital chez les sous-traitants contribuant au projet. Ceci limite ainsi l'utilisation d'approches basées sur l'apprentissage automatique supervisée qui nécessite beaucoup de données de qualité.

Tableau 2.2 Synthèse des articles utilisant l'apprentissage automatique pour la planification dans un cadre de données réelles

| Article | Objectif | Résultat notable |
|---|---|---|
| (Lind, Mary R et al., 2000) | Prédire les dépassements de durée | Meilleur qu'un modèle de régression linéaire |
| (Relich, Marcin et al., 2014) | Prédire la durée du projet | Meilleur qu'une approche statistique<br>Grosse dépendance de la qualité des données |
| {López-Martín, 2015 #170} | Prédire la durée du projet | Meilleur qu'un modèle de régression<br>Difficulté de trouver des projets réels avec suffisamment de données |
| {Sharqi, 2006 #171} | Prédire la durée du projet | Résultat calculé sur seulement 5 projets<br>Difficulté de prendre en compte les variables incertaines |
| {Guo, 2019 #172} | Prédire la durée du projet | Meilleur qu'un modèle de régression linéaire<br>Particulièrement adapté au contexte particulier de la législation chinoise |

2.2.5.3 **Valorisation des données des projets passés**

Certains auteurs utilisent des approches mixtes basées sur l'expérience acquise grâce à l'analyse et à la valorisation des données des projets antérieurs. Cette valorisation consiste à la réutilisation des données afin de les analyser, de les traiter afin d'en obtenir des informations à valeur ajoutée.

C'est le cas de Mikulakova, Eva et al. (2010) qui proposent une approche permettant d'utiliser les projets déjà terminés. À partir de la définition du projet et de l'ensemble des projets présents dans la base de données, le système identifie les caractéristiques du projet semblables à des projets passés. Ensuite, il peut évaluer les différentes alternatives et proposer aux gestionnaires de projet les séquences de construction les plus intéressantes. Ceci est tout à fait pertinent dans le cadre de la gestion et de la mise à jour des échéanciers au fur et à mesure que le projet se définit, mais également pour prévoir les aléas pouvant venir perturber le projet. Contrairement aux approches précédentes qui utilisent l'apprentissage automatique pour déterminer l'importance des caractéristiques du projet, cette approche laisse le choix au gestionnaire de définir les caractéristiques du projet à prendre en compte ainsi que l'importance accordée à ces caractéristiques dans la planification du projet. Ces choix sont basés sur l'expérience du gestionnaire de projet.

## 2.3 Analyse critique

Chacune des approches précédentes apporte des éléments intéressants pour la planification de grands projets de construction quand différentes alternatives de modes de construction sont considérées.

La résolution du problème RCCP, grâce aux approches de planification hiérarchique, peut être transposée pour proposer un échéancier de projet dans les premières phases de projet. Pourtant, bien que ces approches prennent en compte l'incertitude ou les différentes possibilités de chevauchement, elles ne considèrent pas des alternatives aussi différenciantes que les modes de construction.

De plus, les approches qui permettent de résoudre le problème MRCPSP ne sont pas compatibles avec la construction modulaire. En effet, les données entrantes pour résoudre ce problème font l'hypothèse d'un réseau d'activité déjà déterminé, alors que le choix entre une construction classique et une construction modulaire change fondamentalement le réseau d'activités. L'utilisation d'un mode de construction induit un ensemble de lots de travaux propre à ce mode de construction, qui peut différer d'un mode de construction classique sur la durée des lots de travaux, le coût, les ressources nécessaires, les relations de précédence entre les lots de travaux, mais aussi sur la nature même des lots de travaux. Pour une construction modulaire, seul l'assemblage est réalisé sur le site de construction. L'implication sur la planification de projet sur le site de construction est alors très importante comparée à un mode de construction classique où tout est réalisé sur le site de construction, car tous les travaux réalisés en usine se font en temps masqué du point de vue du site de construction.

De la même façon, certaines approches mettent en avant les facteurs de succès pour la construction modulaire. Ces derniers ne sont pas particulièrement axés sur la planification du projet, mais plutôt sur la réussite du projet, c'est-à-dire l'achèvement du projet en respectant les contraintes de coût et de durée définies au début du projet.

En ce qui concerne l'intelligence artificielle, (Smith, 2018) a montré que des séquences très différentes de celles classiquement utilisées pouvaient être proposées par un algorithme de *machine learning*. Malgré tout, cette dernière proposition se concentre uniquement sur l'ordonnancement des pièces à installer et ne considère pas des lots de travaux avec plusieurs modes.

D’autres approches permettent de prédire le coût et la durée, mais elles se basent sur un grand nombre de données issues de projets réels pour obtenir un résultat pertinent. De plus, elles se concentrent sur le projet au complet et non sur les lots de travaux. Ces approches ne permettent donc en aucun cas de proposer un échéancier de projet. De plus, aucune de ces approches n’envisage la possibilité d’avoir plusieurs modes de construction.

L’approche la plus pertinente qui a été relevée dans la littérature est celle de Mikulakova, Eva et al. (2010) qui ont proposé un outil d’aide à la décision basé sur la connaissance acquise, donc sur les projets passés. Cet outil est capable de considérer plusieurs alternatives dans la planification, ce qui est tout à fait pertinent pour la construction modulaire. Cependant, le choix du mode est unique, c’est-à-dire que cet outil ne permet à aucun moment de générer des échéanciers mêlant construction modulaire et construction classique. De plus, les choix sont guidés par le gestionnaire de projet qui décide de l’importance des facteurs à prendre en compte. Cet outil permet alors d’évaluer des scénarios de planification à la suite d’un aléa de projet suivant les priorités et l’expérience du gestionnaire. Il ne peut toutefois pas générer un échéancier de projet en considérant plusieurs modes de construction sur la base des caractéristiques du projet et de ses lots de travail.

## 2.4 Conclusion

Cette revue de littérature a permis de préciser le contexte de la planification de grands projets de construction pour lesquels la planification hiérarchique est très pertinente. Elle a également présenté les spécificités de la construction modulaire face à la construction classique.

Dans le cadre de ces grands projets de construction, la planification au niveau tactique lors des premières phases de définition du projet est un élément critique. Elle l’est encore plus lorsque l’objectif est d’envisager plusieurs modes de construction qui peuvent avoir un impact important sur la planification. L’impact de ce choix n’est pas seulement sur le coût, la durée ou les ressources nécessaires, il peut également être dans le choix même des parties prenantes telles que les fournisseurs.

Parmi l’ensemble des approches recensées dans cette revue de littérature, on constate qu’aucune ne propose un outil permettant de développer des échéanciers de projet au niveau tactique considérant la construction modulaire dans les premières phases de définition de projet, au moment où le choix du mode de construction est critique et que le degré de définition du projet est encore

faible. Le présent mémoire vise à pallier cette limitation. Pour y arriver, la démarche de recherche utilisée est présentée au prochain chapitre.

# CHAPITRE 3 MÉTHODOLOGIE DE RECHERCHE

Ce chapitre présente les objectifs spécifiques de ce projet de recherche. On y présente également les hypothèses de recherche ainsi que la démarche entreprise dans ce travail afin de répondre aux objectifs.

## 3.1 Objectif de recherche

L'analyse critique de la littérature a mis en évidence l'absence de modèle de planification permettant de générer des échéanciers dans les phases préliminaires du projet en prenant en compte différents modes de construction, comme la construction modulaire. De plus, la revue de littérature a permis de mettre en avant les possibilités de l'apprentissage automatique afin de proposer des séquences tout à fait performantes dans un contexte similaire, à travers notamment le travail de Smith (2018) dont le modèle utilise un réseau de neurones permettant de générer des séquences qui planifient l'installation de pièces de construction.

L'objectif principal de cette recherche est donc :

- Montrer la faisabilité d'un outil basé sur l'apprentissage automatique permettant au gestionnaire de projet de planifier un projet de construction en prenant en compte les différents modes de construction tels que la préfabrication.

Cet objectif requiert l'atteinte des sous-objectifs suivants :

- Sous-objectif 1 : Définir un modèle de planification initiale adapté au contexte de la construction modulaire ;
- Sous-objectif 2 : Développer un outil utilisant l'apprentissage automatique générant des échéanciers compatibles avec le modèle défini précédemment ; et
- Sous-objectif 3 : Valider l'applicabilité de l'apprentissage automatique pour cette problématique.

Ce travail de recherche est exploratoire. L'objectif n'est pas d'optimiser les performances du modèle, mais de valider son intérêt.

## 3.2 Hypothèses de travail

Le choix du mode de construction est effectué en pratique dès les prémices du projet, à l'étape d'initialisation, correspondant aux étapes FEL0/1 dans le cadre FEL. La revue de littérature a montré que le mode de construction est choisi en fonction de plusieurs paramètres, dont les caractéristiques générales du projet comme la structure organisationnelle du projet ou la maturité des technologies utilisées, mais surtout l'expérience de l'entreprise et l'expérience du gestionnaire de projet (Li, L. et al., 2018; Song, Jongchul et al., 2005).

Or, lors de ces étapes, l'échéancier sert des enjeux stratégiques et est généralement présenté sous la forme d'un diagramme de Gantt sans relation de précédence. Ces échéanciers présentent les grandes phases du projet, et non les lots de travaux. Ainsi, le choix du mode de construction ne peut se faire que d'un point de vue global. Les phases FEL2 et FEL3 qui permettent de déterminer des échéanciers au niveau tactique et opérationnel ont pour entrée le mode de construction.

Cependant, il pourrait être intéressant de valider le choix du mode de construction lors de l'étape de préfaisabilité (FEL2). Comme présenté dans le Tableau 3.1, on utilise généralement des échéanciers de classe 4 au sens de l'AACE qui supposent un degré de définition du projet compris entre 1 % et 15 %. La fonction de l'édifice à construire est définie dès l'étape de concept. Avec cette définition, on peut raisonnablement penser que les espaces servant l'objectif de l'édifice sont déterminés, sans en connaitre les détails. On supposera dans ce travail que ce degré de définition permet déjà de connaitre le type des grands espaces de l'édifice à construire. Ainsi, on peut considérer dans la planification plusieurs modes de construction pour chacun de ces espaces et évaluer les différentes alternatives d'échéanciers.

Comme mentionné dans la revue de la littérature, la méthode CPM est une technique largement utilisée dans les premières phases du projet par les gestionnaires de projet. L'objectif de cette méthode est de définir un chemin critique dans le réseau d'activité conditionnant le délai final, et ainsi proposer une séquence d'activités. Elle suppose de connaitre la durée de chacune des activités ainsi que les liens de précédence entre celles-ci. Avec cette méthode, on se place également dans un contexte idéal pour lequel les ressources sont illimitées. En pratique, cette hypothèse est difficile à respecter, car les contraintes de ressources représentent un enjeu significatif de la planification. Cependant, les besoins ainsi que les moyens du projet sont définis pendant l'étape de faisabilité avec des échéanciers de classe 3, comme définis par l'AACE. Le contexte de ce travail suppose de

se situer au niveau de l'étape de préfaisabilité, on peut donc raisonnablement supposer des ressources illimitées.

Les deux hypothèses principales sont résumées ainsi :

- Le degré de définition du projet permet de connaitre sommairement les espaces inclus dans l'édifice à construire ; et
- Les ressources sont considérées comme illimitées.

Tableau 3.1 Correspondance entre FEL, classe d'échéanciers et phase de projet

| FEL | Classe d'échéancier AACE | Phase |
|---|---|---|
| FEL0 | Classe 5 | Préconcept |
| FEL1 | Classe 4 | Concept |
| FEL2 | Classe 4 | Préfaisabilité |
| FEL3 | Classe 3 | Faisabilité |
| Non applicable | Classe 2 et Classe 1 | Exécution |

## 3.3 Démarche de recherche

La stratégie de recherche suivie dans ce travail est une adaptation de celle utilisée dans (Cherkaoui, Kaouthar, 2017). Elle est constituée de cinq étapes représentées sur la Figure 3.1. L'étape 1 consiste à définir le modèle de planification adapté à la problématique identifiée ; un outil est ensuite développé à l'étape 2 afin de proposer une solution à ce modèle. Ce dernier ayant besoin d'instances de projet, l'étape 3 consiste à générer ces instances. Enfin, l'outil est testé lors de l'expérimentation à l'étape 4 puis validé à l'étape 5.

Dans le contexte de ce travail de recherche exploratoire, le cadre est défini de manière simple spécifiquement pour permettre de valider l'intérêt de l'outil. Selon les hypothèses définies

précédemment, ce modèle est une adaptation simplifiée du modèle de la méthode CPM en multimode pour la planification initiale.

Les chercheurs préconisent de résoudre ces modèles en proposant d'abord des méthodes de résolution exacte. Cependant, la littérature a montré l'intérêt que pouvait avoir l'utilisation de l'intelligence artificielle dans ces problématiques. On constate dans le cadre qui est défini que le nombre de possibilités peut être très grand même pour des projets de taille réduite. La résolution grâce à une méthode d'intelligence artificielle s'avère alors intéressante.

La deuxième étape consiste à développer un prototype d'outil permettant de générer des échéanciers dans le contexte précisé précédemment. Cet outil est développé en utilisant un algorithme de *machine learning* similaire à celui proposé par Smith (2018). La prise en compte de la construction modulaire augmente considérablement le nombre de possibilités pour la planification. On modélise le processus de planification par analogie avec un jeu dont l'objectif est de trouver la meilleure séquence en choisissant tour à tour le meilleur lot de travail à planifier avec le mode de construction idéal. Une fois entrainé, cet algorithme peut être appliqué à d'autres projets similaires pour obtenir de bonnes performances.

Ce modèle combine un réseau de neurones en parallèle avec un arbre de recherche de Monte-Carlo. Les avantages de ce type d'algorithme sont expliqués au chapitre 4.

Le modèle sera entrainé et testé sur des instances de projet fictives comme le recommande la littérature, étant donné la difficulté à obtenir des données réelles. Ces instances ont pour objectif de reproduire les caractéristiques de projets réels. De plus, l'utilisation d'un algorithme d'intelligence artificielle suppose de disposer d'un très grand nombre de données, nécessitant une structure bien particulière. Dans ce sens, les instances utilisées ici sont construites à partir d'hypothèses définies et justifiées à l'étape 3 de la démarche. Ces hypothèses se basent sur la réalité des projets de construction modulaire.

L'étape 4 est l'expérimentation dont l'objectif est d'évaluer la sensibilité du modèle face à la modification de certains paramètres d'apprentissage détaillés au chapitre 5, comme le nombre d'itérations d'apprentissage, le nombre de parcours de l'arbre et bien d'autres. Les performances du modèle sont mesurées en calculant un score reflétant la qualité de l'échéancier qui est proposée par le modèle. Ce score dépend du coût et de la durée totale de la séquence. Son calcul est présenté en détail dans le chapitre 4.

L'étape 5 correspond à la validation du modèle. Cette validation doit se faire selon plusieurs axes. La performance des planifications proposées par le modèle est évaluée et comparée à deux autres modèles. L'un de ces modèles se base sur un choix aléatoire du lot de travail. Le deuxième est basé sur une règle de priorité qui sélectionne uniquement les lots de travaux avec le mode de construction classique. L'objectif de ce travail n'est pas d'optimiser les performances de ce modèle, mais de valider son intérêt.

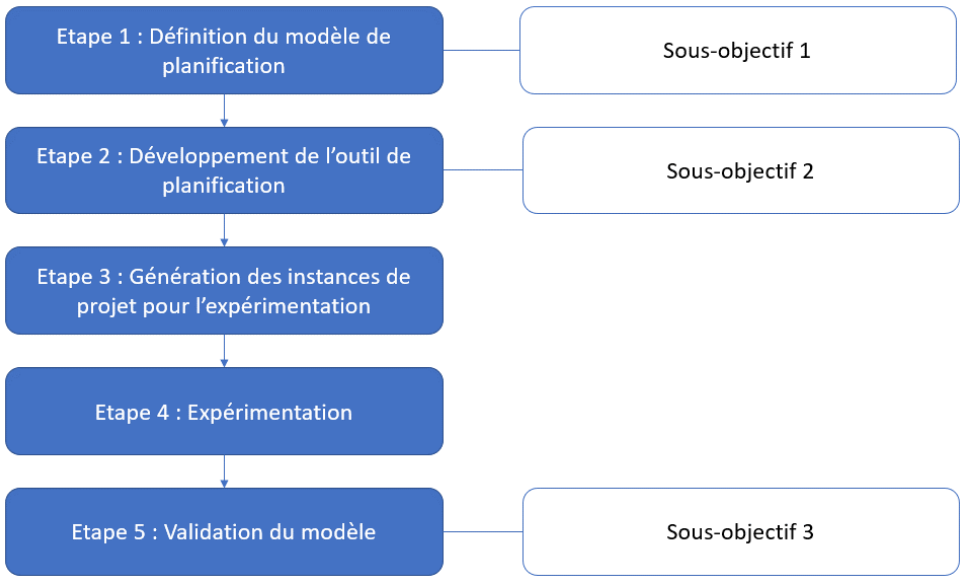


Figure 3.1 Étapes de la méthode de recherche

## 3.4 Conclusion

La méthode de recherche a été présentée dans ce chapitre. Afin de répondre au manque d'outil pour générer des échéanciers dans l'étape de préfaisabilité en considérant la construction modulaire, ce mémoire de recherche propose d'appliquer un algorithme de *machine learning* basé sur le modèle d'AlphaGo. Dans le chapitre suivant, nous expliquons en détail la modélisation de la planification de projet retenue ainsi que le développement de l'outil.

# CHAPITRE 4 PROPOSITION DU MODÈLE

Ce chapitre a pour but de présenter et justifier le modèle permettant de proposer un échéancier de projet optimisé en prenant en compte les différents modes de construction. Dans ce chapitre, le problème est défini par analogie avec la théorie des jeux adaptée à la problématique de ce travail de recherche. En effet, la revue de littérature a montré qu'aucun modèle ne permettait de décrire convenablement le problème de planification initiale de projet de construction avec plusieurs modes de construction. Dans un second temps, le modèle de résolution est décrit de manière détaillée. Celui-ci est basé sur la combinaison d'un arbre de recherche de Monte-Carlo et d'un réseau de neurones.

## 4.1 Rapprochement avec la théorie des jeux

### 4.1.1 Justification du modèle

La revue de littérature a permis de permettre en avant le travail de Smith (2018) qui œuvre dans un contexte très similaire. En effet, son objectif est de placer dans l'espace et dans le temps des pièces de construction afin de trouver la meilleure séquence d'installation. Pour ce faire, Smith (2018) propose de faire une analogie avec la théorie des jeux et plus particulièrement avec les jeux de plateaux comme le jeu d'échecs ou le jeu de Go.

En théorie des jeux, on distingue trois critères afin de déterminer la nature du jeu qui sont le caractère statique ou dynamique du jeu, la nature de l'information et la capacité des joueurs à s'engager sur leurs décisions (Lepelley, Dominique et al., 2013). Le jeu est dynamique quand il donne de l'information à au moins un joueur, sinon il est statique. Le deuxième critère permet de différencier les jeux coopératifs et non coopératifs. Dans l'approche coopérative, les décisions sont prises en commun et peuvent alors faire intervenir une étape de négociation. Avec l'approche non coopérative, chaque joueur agit librement dans son propre intérêt. Enfin, la nature de l'information peut être parfaite ou imparfaite. Dans le cadre de jeux à information parfaite, l'ensemble des joueurs connaissent toutes les informations nécessaires pour leur permettre de prendre la meilleure décision, ce qui diffère des jeux à information imparfaite pour lesquels certains joueurs peuvent avoir des informations complémentaires au moment de la prise de décision.

L'analogie proposée par Smith (2018) consiste à rapprocher le problème de planification avec un jeu de plateau comme les échecs qui est un jeu statique, non coopératif et à information parfaite. En effet, on peut considérer le problème de planification comme un jeu pour lequel le plateau est l'espace dans lequel les pièces doivent être installées. Chaque mouvement de pièce du jeu correspond à un choix d'une pièce à installer. On choisit alors tour après tour une pièce à installer dans l'état de planification qui correspond à l'état de jeu et qui représente l'emplacement de chacune des pièces ainsi que les pièces jouées précédemment.

De cette façon, Smith (2018) peut adapter les algorithmes utilisés pour les jeux de ce type, comme AlphaGo, en transformant ce jeu avec un joueur unique. Ainsi, l'objectif n'est pas simplement de gagner contre son adversaire, mais plutôt contre soi-même en obtenant le meilleur score. Ce score correspond à une fonction qui caractérise le jeu.

Pour les jeux à information parfaite, cette fonction a toujours une valeur optimale. Ces jeux peuvent alors être résolus en calculant de manière récursive cette fonction pour trouver l'optimum dans un arbre de recherche. Mais cet arbre peut présenter un très grand nombre de possibilités, comme environ $250^{150}$ pour le jeu de Go et $35^{80}$ pour les échecs. Ainsi, l'évaluation exhaustive des possibilités est impossible (Van Den Herik, H Jaap et al., 2002).

L'évaluation du nombre de possibilités est calculée approximativement avec la formule $b^d$ où $b$ est le nombre de coups possible à chaque tour et $d$ le nombre approximatif de coups pour terminer la partie. Dans notre cas, on peut considérer que le nombre de modules possibles à planifier dépend de la taille de l'édifice, tout comme le nombre d'itérations permettant de planifier tous les modules. Ce travail de recherche propose un modèle de planification pour les grands de projets de construction et d'ingénierie, on peut alors raisonnablement supposer que le nombre de modules à planifier est suffisamment grand, rendant la recherche exhaustive impossible.

La stratégie utilisée afin de résoudre ces problèmes consiste alors à (1) réduire la taille de l'arbre de recherche en considérant à chaque itération un sous arbre dont la racine représente l'état de jeu en cours et (2) de réduire le nombre de coups possibles en calculant des probabilités pour chacun des coups. Par exemple, les arbres de recherche de Monte-Carlo sont tout à fait pertinents, car chaque itération de l'algorithme permet de parcourir toute la profondeur de l'arbre afin d'obtenir la valeur de l'état final (Coulom, Rémi, 2006). Ainsi, il construit les probabilités pour les branches disponibles comme expliqué dans la revue de littérature. À chaque itération, les probabilités sont

affinées et l'algorithme converge alors de manière asymptotique vers l'optimum de la fonction. On observe que pour la théorie des jeux, des algorithmes très performants combinent un arbre de recherche de Monte-Carlo et l'aide humaine afin d'ajuster les probabilités. Le travail de Silver, David et al. (2016) propose avec AlphaGo de remplacer cette aide humaine par un réseau de neurones, permettant de proposer encore plus d'itérations. Cet algorithme et ses variantes sont les méthodes de résolution les plus performantes à l'heure actuelle.

Tout comme dans (Silver, David et al., 2017; Smith, 2018), le modèle développé ici est alors une combinaison entre un arbre de recherche de Monte-Carlo et un réseau de neurones.

### 4.1.2 Analogie

Nous faisons ici une analogie similaire à Smith (2018) en considérant notre problème comme un jeu de plateau statique en information parfaite avec un joueur pour lequel l'objectif est de choisir tour après tour un module à planifier avec un certain mode afin d'obtenir le meilleur échéancier dont la performance est le résultat d'une fonction présentée plus tard.

Le plateau de jeu est l'espace du site de construction découpé en cubes élémentaires formant un grand cube en trois dimensions correspondant à la taille de l'édifice à construire. Selon les hypothèses de ce travail énoncées au chapitre précédent, l'édifice est supposé partiellement défini. Cette définition inclut la spécification de chacun des grands espaces de l'édifice. Par exemple, on saura où se trouvent les parties dédiées aux logements, celles dédiées aux espaces de réception, les cuisines … Chacun de ces espaces aura alors des besoins particuliers pour la construction comme les besoins d'alimentation en eau ou en électricité ainsi que le niveau d'exigences. Sans en connaitre les détails, on peut alors tout de même évaluer la durée et les coûts approximatifs de la construction de ces espaces.

Dans notre travail, chacun de ces espaces est appelé un *module*. On a la possibilité de construire chaque *module* suivant plusieurs *modes* de construction. Suivant la fonction du module dans l'édifice, appelée *type*, le coût et la durée de construction d'un module dépendent du mode de construction. Les différents types de modules, les modes de construction ainsi que la durée et les coûts associés sont des paramètres d'entrée du modèle présentés en détail au paragraphe 5.1.2.

Au début du jeu, le joueur connait l'ensemble des modules à installer avec leur emplacement ainsi que les modes de construction possibles, associés à leur coût et à leur durée correspondante.

L'objectif du jeu est alors de choisir tour après tour un module à planifier avec son mode de construction de façon à minimiser le résultat d'une fonction dépendant du coût et de la durée. À chaque tour, le joueur a le droit de planifier un ou plusieurs modules, ou de ne rien faire. Chaque tour correspond à un jour, en commençant par le premier jour de construction sur le site. Le dernier tour correspond alors au dernier jour de construction.

Ce jeu rassemble quelques règles supplémentaires.

Tout d'abord, lorsqu'un module est planifié, l'espace où ce module doit être installé est indisponible pendant toute la durée d'installation du module. De plus, afin de simuler de manière plus réaliste le mode de construction modulaire qui nécessite très souvent l'utilisation d'une grue, certains modes de construction occupent également un second espace qui reste également indisponible pendant toute la durée d'installation du module. Ce second espace doit être adjacent à l'espace où le module est installé et est choisi par le joueur. Naturellement, cet espace doit être disponible afin de pouvoir planifier un module le nécessitant.

De plus, il est à noter qu'un module ne peut être planifié seulement-ci les modules se situant sous lui ont déjà planifiés. Enfin, une fois qu'un module est planifié, il ne peut être annulé ni même changer de mode de construction.

## 4.2 Description du modèle

### 4.2.1 Description globale

Le modèle développé pour répondre à la problématique est composé d'une combinaison d'un arbre de recherche de Monte-Carlo avec un réseau de neurones, tel que représenté sur la Figure 4.1.

Un état de planification est un échéancier partiel qui décrit l'ensemble des modules déjà planifiés et restant à planifier avec leurs caractéristiques. Dans l'algorithme, un état est matérialisé par une matrice de taille NxM où N est le nombre de lots de travail, et M est le nombre de caractéristiques de chaque lot.

Les caractéristiques considérées sont :

- Type de module ;

- Mode de construction ;
- Coût ;
- Durée ;
- Emplacement : 3 coordonnées x, y et z ;
- Possibilité de planification ; et
- Date de début.

Le type de module et l'emplacement permettent de connaitre le module à planifier. Comme vu précédemment, on considère qu'un lot de travail correspond à l'installation d'un module selon un mode de construction particulier. Les différents types de modules permettent de décrire les espaces de l'édifice qui sont définis partiellement avant la planification. Parmi ces types de modules, on peut trouver par exemple des espaces de logements, des espaces de déplacement ou bien encore des espaces plus spécifiques comme des sanitaires. Chacun de ces espaces a des besoins particuliers pour sa construction. Dans le modèle, le type de module est présenté sous la forme d'un entier qui l'identifie.

Le mode de construction correspond à la façon dont le mode est construit et installé. Par exemple, on peut avoir le mode classique, le mode modulaire, le mode panel. De la même façon, un mode de construction est représenté par un entier qui l'identifie.

La durée d'un module correspond à la durée en jours nécessaire pour construire et installer le module. Ainsi, l'emplacement sur lequel est installé le module est occupé pendant toute la durée du lot de travail.

La durée et le coût sont directement liés au type du module et au mode de construction. En entrée du modèle, on définit des durées et des coûts de référence pour chaque type de module construit selon les différents modes de construction. Ces informations sont déterminées approximativement selon les pratiques dans l'industrie de la construction ; le gestionnaire pourra alors ajuster ces valeurs afin de correspondre au mieux à la réalité de son entreprise. Cette définition sommaire est cohérente, car à ce moment du projet, la définition de l'édifice est encore sommaire ; il est alors très compliqué de proposer des coûts et durées précises pour chacun des lots de travaux

spécifiquement. De plus, cette solution permet de générer un maximum de données d'entrainement, car le modèle se base directement sur les coûts et durées de référence. Ceci est essentiel pour assurer l'apprentissage du modèle.

L'emplacement du module est décrit par 3 entiers x, y et z qui représentent les 3 coordonnées dans l'espace. En effet, l'espace est découpé en cubes dans lesquels les modules sont définis. On considère qu'un module doit être installé dans un cube. Tout comme les coûts et les durées, l'emplacement des modules est une entrée dans mon modèle, selon la définition préalable de l'édifice.

La possibilité de planification est une variable mise à jour à chaque changement d'état de planification. Elle est calculée en fonction des dépendances entre les lots de travail ainsi que des lots de travail déjà planifiés. Si le module correspondant a déjà été planifié selon un autre mode de construction, elle vaut -1. Sinon, elle vaut 0 si les dépendances entre les lots de travail ne permettent pas de le planifier, et 1 si ce lot de travail peut être planifié.

La date de début correspond à la date à laquelle le module commence à être construit dans l'échéancier. Cette date est calculée par le modèle et permet de construire l'échéancier proposé.

L'arbre de recherche décrit l'ensemble des possibilités d'échéanciers à partir d'un état de planification. Chaque nœud ou feuille de l'arbre de recherche correspond à un état de planification. Chaque branche correspond quant à elle à un lot de travail à planifier, appelé aussi action. Ainsi, à partir d'un nœud, les branches connectées à ce nœud représentent les lots de travail possibles à planifier en prenant en compte les dépendances entre les lots.

Pour parcourir l'arbre, le réseau de neurones calcule la valeur probable de l'état de planification ainsi que les probabilités de planifier les prochains lots de travail. Ce réseau de neurones est tout d'abord entrainé grâce à un ensemble de données d'entrainement générées par un autre algorithme décrit au paragraphe 5.1.2. Une fois entrainé, le modèle complet peut alors être utilisé pour générer des échéanciers de projet. Le réseau de neurones et sa logique d'entrainement sont décrits au paragraphe 4.2.3.

La valeur probable d'un état de planification est calculée grâce au réseau de neurones. Le calcul de cette valeur prend alors en compte les modules déjà planifiés et restants à planifier et permet de mesurer l'intérêt de cet état de planification.

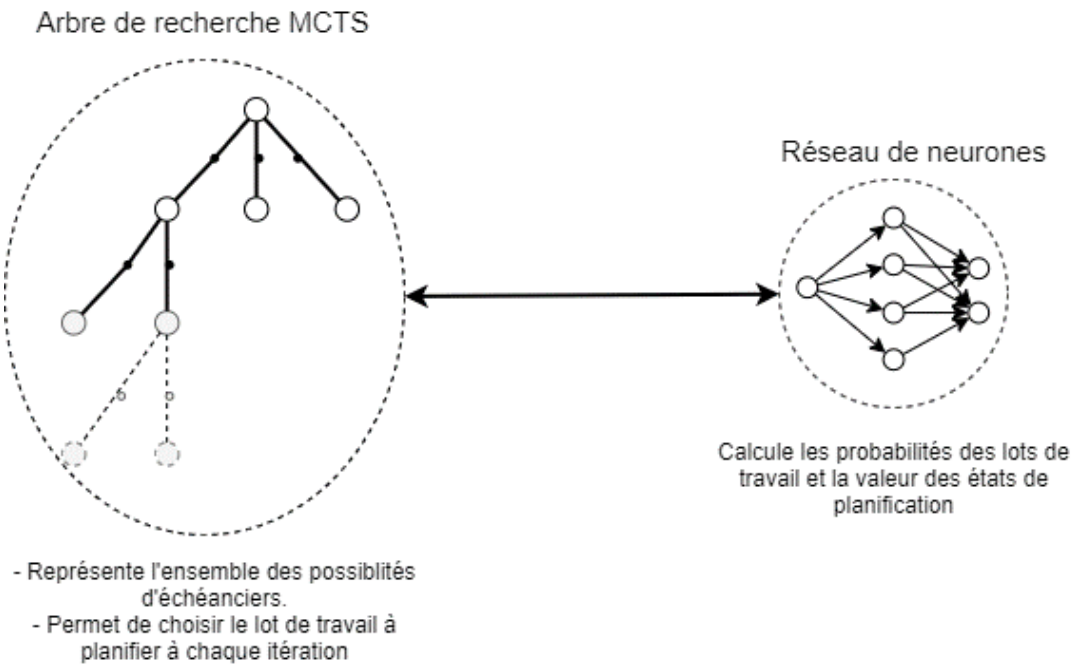


Figure 4.1 Modèle global

Le modèle présenté ici est basé sur un algorithme itératif dans lequel chaque itération permet de choisir un module à planifier, et donc de passer d'un état de planification au suivant, jusqu'à ce que l'échéancier soit complet.

À chaque itération, l'arbre de recherche permet alors de fournir les probabilités de planifier les prochains modules, puis choisit le module ayant la plus forte probabilité. Pour déterminer ces probabilités à part d'un état de planification, l'arbre de recherche est aidé par le réseau de neurones et suit un cycle de quatre étapes *k* fois, *k* étant un paramètre de l'algorithme.

Une fois ce cycle effectué *k* fois, on choisit l'action avec la plus forte probabilité, ce qui correspond à choisir le lot de travail à planifier. Ceci représente alors une itération. On reproduit cette logique le nombre de fois nécessaire pour générer l'échéancier complet.

### 4.2.2 Arbre de recherche de Monte Carlo

Un arbre de recherche de Monte Carlo est un algorithme de recherche heuristique généralement utilisé pour la prise de décision. L'arbre représente l'ensemble des possibles, ici, l'ensemble des échéanciers possibles pour un projet de construction à partir d'un état de planification.

Essentiellement, une action, ou une branche, est caractérisée par quatre valeurs : *N*, *W*, *Q* et *P*.

*N* est le nombre de fois où la branche a été choisie à partir du nœud précédent. *W* est la valeur totale du nœud suivant. *Q* est la valeur moyenne du nœud suivant. *P* est la probabilité de choisir la branche.

Lors d'une itération de l'algorithme, on choisit un lot de travail à planifier. Pour choisir ce lot, le modèle détermine les probabilités des lots de travaux possibles. Pour obtenir ces probabilités, l'arbre de recherche est parcouru *p* fois en suivant les 4 étapes représentées sur la Figure 3.1 : la sélection, l'expansion, la simulation et la rétropropagation.

Tout d'abord, il s'agit de l'étape de sélection pour laquelle l'arbre de recherche est parcouru jusqu'à une feuille en choisissant à chaque fois les branches en suivant une fonction de choix mêlant l'exploration et l'exploitation de l'arbre. L'exploration de l'arbre suppose de choisir volontairement des branches qui ont a priori une probabilité plus faible d'aboutir à un échéancier intéressant. Elle évite alors de se restreindre aux optimums locaux. L'exploitation permet au contraire de choisir des branches avec une forte probabilité pour confirmer ou non l'intérêt de cette branche.

A chaque nœud *s*, l'arbre choisit la branche *a* qui maximise la valeur *Q* + *U*. *Q* est la valeur moyenne de l'état planification suivant calculée en divisant *W* par *N*. *Q* représente alors l'exploitation de l'arbre. *U* est une fonction de *P* et *N* qui augmente quand la branche n'a pas été souvent choisie. *U* représente donc l'exploration de l'arbre.

$$Q(s,a) = \frac{W(s,a)}{N(s,a)} \tag{1}$$

$$U = f(P,N) = c_{puct}.P(s,a).\frac{\sqrt{N(s,a)}}{1+N(s,a)} \tag{2}$$

Cette fonction est une variante de celle utilisée dans l'algorithme PUCT où $c_{puct}$ est une constante définissant le niveau d'exploration (Rosin, Christopher D, 2011).

Lors des premières itérations, les valeurs *W* sont petites, c'est donc l'exploration qui est privilégiée. Puis le terme *Q* augmente et l'exploitation est privilégiée. On s'assure alors d'identifier les branches menant aux échéanciers les plus intéressants, et les branches les plus visitées seront les meilleures branches (Rosin, Christopher D, 2011; Silver, David et al., 2016).

Une fois une feuille atteinte, l'algorithme procède à la seconde étape du cycle, qui est l'expansion. Cette étape consiste à développer l'arbre en identifiant les nouvelles feuilles suivantes dans le cas où la feuille n'est pas une feuille finale qui correspond à un état de planification pour un échéancier complet. Les feuilles sont alors initialisées avec :

$$N = 0 \tag{3}$$

$$W = 0 \tag{4}$$

$$Q = 0 \tag{5}$$

$$P = f(s) \tag{6}$$

où $f$ *est la fonction représentant le réseau de neurones et* $s$ *est un noeud*

Ensuite, l'étape de simulation permet de déterminer la valeur de la feuille. Cette étape est réalisée par le réseau de neurones qui fournit la valeur *v* de l'état de planification. Cette valeur représente l'intérêt de ce nœud. Plus la valeur est grande, plus le nœud a des chances de mener à des échéanciers performants, c'est-à-dire avec un coût et une durée faible.

Enfin, l'étape de rétropropagation permet de mettre à jour les valeurs de chaque branche. Ainsi, pour chacune des branches parcourues dans l'itération en cours, *N* est incrémenté de 1, on ajoute à *W* la valeur *v* de l'état et *Q* est recalculé en fonction des nouvelles valeurs de *N* et *W*.

$$N \rightarrow N + 1 \tag{7}$$

$$W \rightarrow W + v \tag{8}$$

$$Q = \frac{W}{N} \tag{9}$$

L'itération est alors terminée. Elle permet notamment de mettre à jour *N* pour les branches de l'arbre.

Ensuite, $k$ itérations de l'arbre sont effectuées. Ainsi, plus l'arbre est parcouru, plus on est capable d'identifier la meilleure branche à choisir. Après ces $k$ itérations, la branche avec la valeur $N$ la plus grande qui sera choisie correspond à la branche la plus visitée.

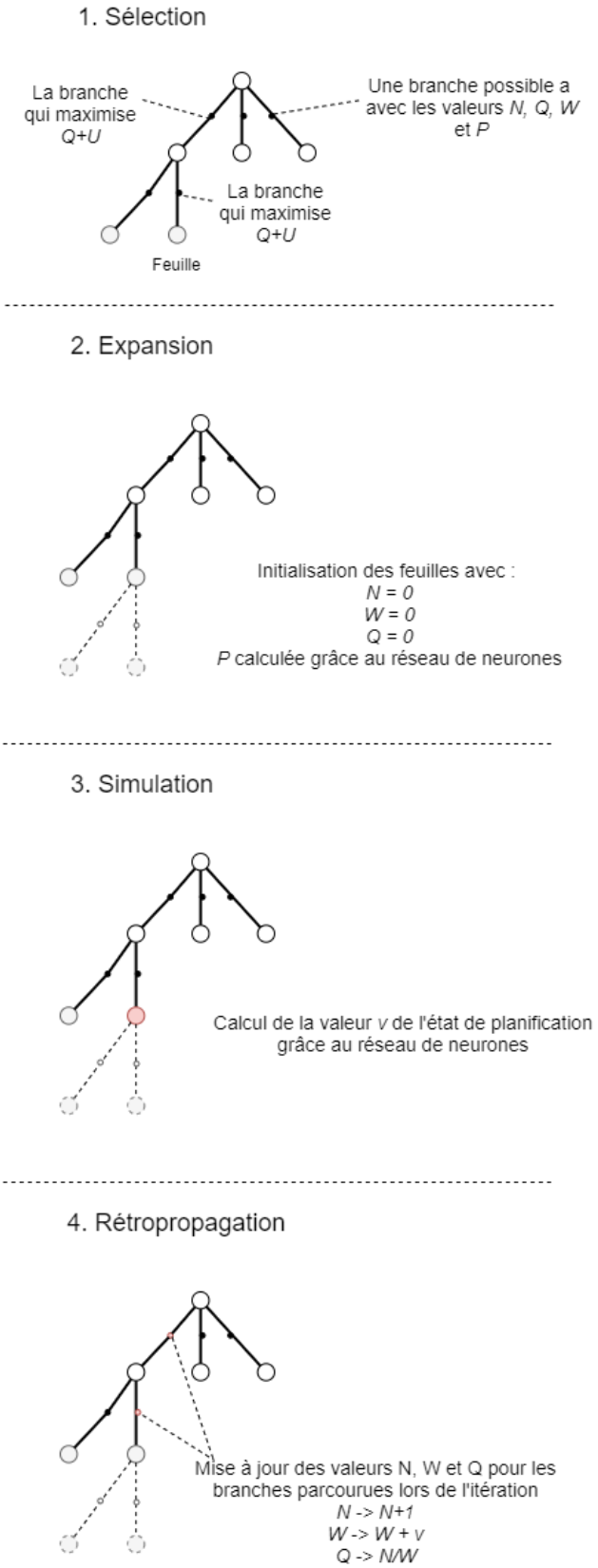


Figure 4.2 Une itération du MCTS

### 4.2.3 Réseau de neurones

#### 4.2.3.1 **Apprentissage automatique**

L'apprentissage automatique, ou *Machine learning*, est un sous domaine de l'intelligence artificielle qui consiste à développer des algorithmes statistiques résolvant des problèmes en apprenant à partir d'exemples. Ces exemples peuvent être observés et rassemblés, ou alors générés par un autre algorithme. L'apprentissage peut être supervisé, semi supervisé, non supervisé ou par renforcement (Burkov, Andriy, 2019).

Pour l'apprentissage supervisé, les données d'entrainement sont des exemples labelisés sous la forme (xi, yi) pour lesquels xi est un vecteur de caractéristiques. Ces caractéristiques sont les données d'entrée de l'algorithme. yi est le label de l'exemple ; il peut faire partie d'un ensemble fini ou être une valeur réelle, voire un élément plus complexe. Ainsi, lors d'un apprentissage supervisé, l'objectif est de créer un modèle permettant de déduire le label yi à partir du vecteur caractéristique xi. Un exemple est de considérer le vecteur xi comme les caractéristiques d'une personne permettant de déduire le genre de cette personne, qui serait alors le label yi. Pour construire ce modèle, des données d'entrainement labellisées sont fournies au modèle et pour chacune d'entre elles, l'algorithme compare la donnée de sortie au label fourni. Cette comparaison permet de mettre à jour le modèle pour que celui se rapproche du label, c'est ainsi qu'on dit que le modèle apprend. Une fois le modèle entrainé, il peut être utilisé afin de définir les labels d'autres vecteurs caractéristiques non labelisés(Burkov, Andriy, 2019).

Dans l'apprentissage non supervisé, les exemples sont non labelisés. La logique est alors d'entrainer le modèle à partir du vecteur de caractéristiques afin de le transformer en un autre vecteur ou une valeur permettant de résoudre une problématique. Les applications classiques concernent la classification (ou *clustering*) pour lequel le modèle peut déterminer à quel groupe de données le vecteur caractéristiques appartient, ou alors la réduction de dimension pour laquelle le modèle donne en sortie un autre vecteur caractéristiques avec une dimension plus faible(Alpaydin, Ethem, 2020).

L'apprentissage semi supervisé est basé sur des exemples labelisés et non labelisés. L'objectif de ce genre de modèle est le même que pour l'apprentissage supervisé. En général, le nombre d'exemples non labelisés est pourtant bien supérieur au nombre d'exemples labelisés. L'avantage

de ce type d'apprentissage est que les exemples non labelisés ajoutent de la complexité au modèle ce qui peut le rendre d'autant plus proche de la problématique et donc apporter des résultats plus pertinents.

Le dernier type d'apprentissage est l'apprentissage par renforcement qui a été le choix réalisé dans le cadre de ce travail de recherche. Pour ce type d'apprentissage, le modèle est dans un environnement dans lequel il peut faire des actions pour chaque état. Dans cet environnement, le modèle est capable de définir chaque état sous la forme d'un vecteur de caractéristiques. Le modèle peut exécuter des actions à chaque état. Chacune des actions peut apporter des récompenses, voire changer l'état. L'objectif ce modèle est alors de construire une politique de choix d'action permettant de maximiser la récompense. Cette politique est une fonction prenant en entrée le vecteur caractéristique et proposant une action en sortie (Sutton, Richard S et al., 2018).

Dans le cadre de ce travail, chaque état correspond à un état de planification comme défini plutôt. Chaque action correspond à la planification d'un module.

Pour ces différents types d'apprentissage, on peut utiliser différents types d'algorithmes. Le choix s'est porté ici sur un réseau de neurones. Un réseau de neurones est un type d'algorithme inspiré du fonctionnement biologique du cerveau animal. Un réseau de neurones est généralement composé d'une succession de couches dont chacune prend pour entrées les sorties de la couche précédente. Les neurones d'une couche sont reliés à ceux d'une autre couche grâce à des synapses auxquelles est associé un poids. Comme illustré dans la Figure 4.3, un neurone assure une fonction de combinaison qui est la somme pondérée des valeurs de sorties des neurones reliés à celui-ci et d'une fonction d'activation *f*. Dans le cadre d'un apprentissage par renforcement, le réseau de neurones est d'abord entrainé comme pour un apprentissage supervisé avec des valeurs de sortie attendues. Ces valeurs de sortie sont représentatives du comportement attendu pour le réseau de neurones afin d'adopter une politique de choix le récompensant. Les valeurs attendues et les valeurs de sortie sont comparées grâce à une fonction de perte. Les poids des synapses évoluent toujours de manière à faire diminuer la fonction de perte. Une fois entrainé, le réseau de neurones propose un nouveau comportement qui devrait être plus performant. Une bonne pratique est de comparer le réseau de neurones entrainé avec sa version précédente sur des situations connues afin de garder seulement la version la plus performante.

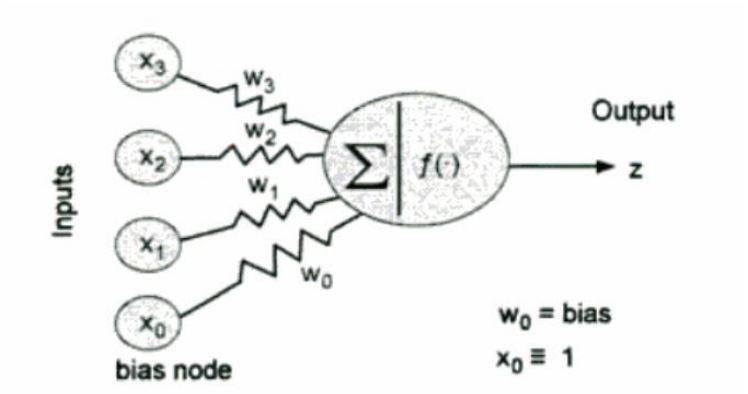


Figure 4.3 Schéma d'un neurone dans un réseau de neurones (Priddy, Kevin L et al., 2005)

4.2.3.2 **Définition du réseau de neurones**

Dans le cadre de ce travail de recherche, l'entrée du réseau de neurones est un vecteur de taille NxM représentant un état de planification pour lequel N est le nombre de modules du problème et M le nombre de caractéristiques.

Le réseau de neurones est ensuite composé d'une couche intermédiaire dense avec 64 neurones et une fonction d'activation de type ReLu pour Rectified Linear Unit. La fonction d'activation a pour formule mathématique :

$$\varphi(net_j) = \max(net_j, 0) \tag{10}$$

$$O\text{ù } net_j = \sum_i w_{i,j}.x_i$$

$w_{ij}$ sont les poids associés aux synapses qui relient les neurones et $x_i$ les valeurs entrantes dans la couche.

Une couche Dense signifie que tous les neurones de la couche sont connectés à tous les neurones de la couche précédente et de la suivante.

L'objectif de ce travail étant de valider l'intérêt de ce type d'outil, le choix de la structure a été fait de manière très simple. Il est recommandé dans les bonnes pratiques de l'apprentissage automatique de toujours commencer avec des structures de réseaux de neurones simples afin d'obtenir un premier résultat, qui pourra ensuite être optimisé (Burkov, Andriy, 2019).

Dans la pratique, les couches Denses couplées à l'utilisation de la fonction d'activation ReLu constituent une structure intéressante pour obtenir un premier résultat. La structure du réseau de neurones est illustrée sur la Figure 4.4.

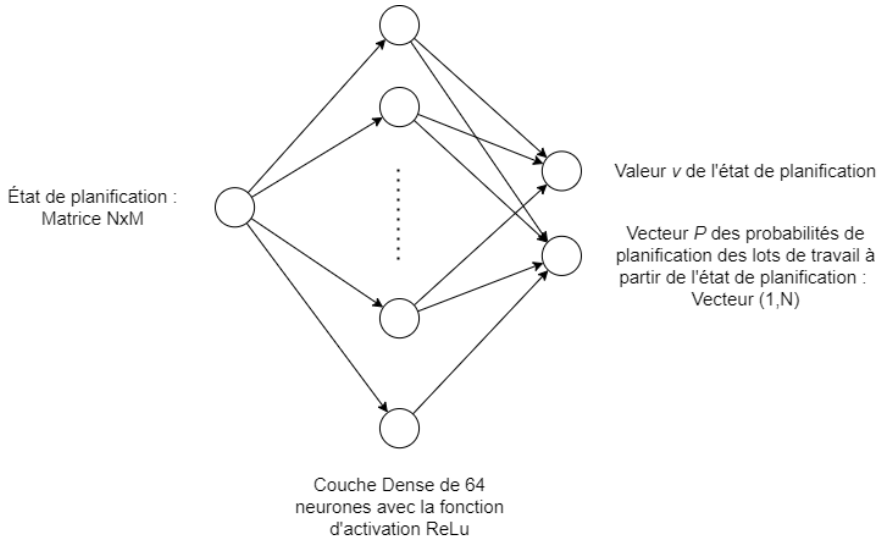


Figure 4.4 Structure du réseau de neurones

#### 4.2.3.3 **Apprentissage par renforcement**

Dans l'apprentissage par renforcement, on dit que l'algorithme vit dans un environnement où il peut évoluer grâce à des actions. Ici, l'environnement dans lequel l'algorithme évolue est la planification d'édifice. Chacune de ces actions peut le faire changer d'état (Burkov, Andriy, 2019).

La particularité de l'apprentissage par renforcement est que l'algorithme apprend de lui-même. Dans notre cas, l'apprentissage est fait par itération. A chaque itération, le réseau de neurones est comparé à une autre version de lui-même. On nomme *réseau de neurones optimal* la version du réseau de neurones utilisée au début de l'itération. En effet, on garde toujours la version la plus performante du réseau de neurones. On nomme également *réseau de neurones temporaire* la version du réseau de neurones entrainée lors de l'itération. Ces deux versions du réseau de neurones ont exactement la même structure, seuls les poids du réseau changent.

À chaque itération, on fournit au *réseau de neurones temporaire* un ensemble d'états de planification afin qu'il calcule les probabilités et les valeurs de ces états de planification, cet ensemble est appelé ensemble d'entrainement. Cet ensemble d'entrainement est généré grâce au MCTS en utilisant *le réseau de neurones optimal*. En effet, à chaque fois que le MCTS est parcouru,

un certain nombre de nœuds sont visités. Chacun de ces nœuds est un état de planifications. Grâce à plusieurs itérations, on stocke l'ensemble de ces états de planification pour former l'ensemble d'entrainement.

Ensuite, on compare *le réseau de neurones temporaire* et *le réseau de neurones optimal.* Pour se faire, on considère une instance de projet dont on propose un échéancier avec chacune des versions du réseau de neurones. La performance de chacun de ces échéanciers est évaluée et comparée. On répète l'opération *n* fois afin de comparer les versions sur un échantillon de projet plus important. On note $n_t$ le nombre de fois où *le réseau de neurones temporaire* est plus performant que *le réseau de neurones optimal.*

*n* est un paramètres d'apprentissage pour le modèle, il correspond au nombre de fois où les deux versions du réseau de neurones sont comparées.

Si $n_t/n$ dépasse une valeur seuil *u* supérieur à *0.5*, on considère que *le réseau de neurones temporaire* est plus performant que *le réseau de neurones optimal.* Dans ce cas, *le réseau de neurones temporaire* devient *le réseau de neurones optimal*. On utilisera cette version du réseau pour la prochaine itération.

Si $n_t/n$ est inférieur à *u, le réseau de neurones optimal* ne change pas.

Cette valeur seuil *u* est un paramètre d'apprentissage pour le modèle. Ce seuil correspond au pourcentage de victoires obtenues par le réseau de neurones temporaire face au réseau de neurones optimal, une victoire correspondant à un échéancier plus performant sur la même instance de projet. *u* doit donc être supérieur à 0.5 afin de s'assurer que le réseau de neurones s'améliore.

Cette itération est alors répétée *i* fois pour permettre au réseau de neurones d'apprendre. On garde finalement *le réseau de neurones optimal* de la dernière itération. *i* est également un paramètre d'apprentissage de l'algorithme. On peut penser que plus il y a d'itérations, plus le réseau de neurones est performant. Dans le prochain chapitre, l'expérimentation montre certaines limites de cette affirmation.

#### 4.2.3.4 **Processus d'entrainement du modèle**

Le processus d'entrainement du modèle est présenté dans la Figure 4.5. Tout d'abord, les premières opérations permettent de remplir l'ensemble entrainement. Pour cela, on commence par considérer une instance de projet générée dans notre cas par l'algorithme présenté au chapitre 5. Ensuite,

l'arbre de recherche est initialisé avec l'instance de projet et en chargeant *le réseau de neurones optimal*. La prochaine étape consiste à faire des simulations qui sont concrètement la recherche d'un échéancier pour cette instance de projet. Lors de ces simulations, tous les états de planification sont conservés. Ils sont ajoutés à la banque de données d'entrainement. On répète ces étapes pour *numEps* instances de projet tel que paramétré.

Ensuite, le réseau de neurones est entrainé avec la banque d'entrainement alimentée précédemment. Une fois entrainé, ce réseau de neurones devient *le réseau de neurones temporaire*. Il est alors comparé au *réseau de neurones optimal* en comparant leur score sur les mêmes instances de projet. On garde alors la meilleure des versions du réseau de neurones qui devient le *nouveau réseau de neurones optimal*. Ceci conclut une itération d'entrainement. Le modèle répète ces opérations *numIters* fois, qui est un paramètre d'apprentissage.

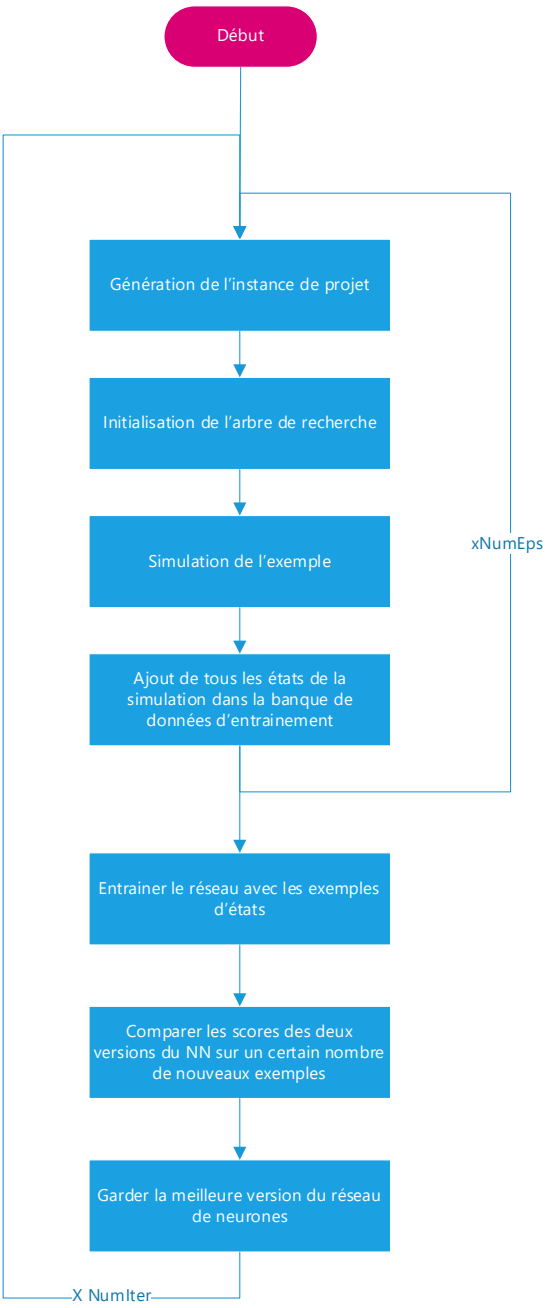


Figure 4.5 Processus d'entrainement du modèle

# CHAPITRE 5 EXPÉRIMENTATION

Ce chapitre présente les dernières étapes de la démarche de recherche, soient l'expérimentation et la validation du modèle. Dans cette phase, nous allons d'abord préciser le cadre expérimental dans lequel le modèle est testé. Ensuite, nous expliquons comment les instances de projet sont générées avant d'expérimenter le modèle grâce à une analyse de sensibilité face à différents paramètres d'apprentissage, ainsi que la comparaison avec deux autres modèles afin de valider l'intérêt du modèle.

## 5.1 Application du modèle

### 5.1.1 Paramètres du modèle

Dans cette section sont présentés les différents paramètres utilisés dans le modèle. Certains de ces paramètres ont fait l'objet d'une partie de l'expérimentation afin de mesurer l'influence de leur variation sur les résultats du modèle. Ils sont résumés dans le

Tableau 5.1 ci-dessous.

Le paramètre Taille est un vecteur de 3 valeurs représentant la taille de l'édifice à construire. Ce paramètre détermine alors la taille de l'espace dans lequel les modules sont planifiés. Ce paramètre est particulièrement utile afin de générer les données d'entrainement, pour fixer la taille des différents éléments dans l'algorithme, comme les matrices représentant les états de planification, mais également pour paramétrer le réseau de neurones.

Tableau 5.1 Description des paramètres du modèle

| Paramètres | Description |
|---|---|
| $Taille = [H, l, L]$ | Vecteur de 3 valeurs représentant la taille de l'édifice à construire. *H* est la hauteur, *l* la largeur et *L* la longueur |
| *numIters* | Nombre d'itérations du modèle |
| *numEps* | Nombre d'états de planification à générer au départ de chaque itération du modèle |
| *updateThreshold* | Seuil en pourcentage minimum à partir duquel le réseau de neurones est mis à jour avec les poids du *réseau de neurones temporaire* |
| *numMCTSSims* | Nombre de simulations de l'arbre de recherche de Monte-Carlo avant de choisir un module à planifier |
| *MaxlenofQueue* | Nombre maximum d'états de planification dans l'ensemble d'entrainement |
| *arenaCompare* | Nombre de confrontations entre le *réseau de neurones optimal* et le *réseau de neurones temporaire* |
| *F* | Fonction d'évaluation du score d'un échéancier |

### 5.1.2 Génération des instances de projet

La présentation du modèle au chapitre 4 a montré la nécessité de générer des instances de projet. Celles-ci servent à entrainer le modèle, mais également à le tester.

Un algorithme a donc été conçu afin de générer un maximum d'instances facilement. Comme précisé précédemment, l'objectif du modèle est de proposer une planification initiale d'un édifice dont on suppose que les grands espaces à construire sont déjà définis. Ces espaces, appelés modules, peuvent être construits en suivant plusieurs modes de construction.

En entrée de cet algorithme, nous avons défini des paramètres présentés dans le Tableau 5.2 qui permettent de générer un ensemble d'instances basées sur ces paramètres.

Tableau 5.2 Paramètres pour la génération d'instances

| Paramètres | Description |
|---|---|
| $Taille = [H, l, L]$ | Vecteur de 3 valeurs représentant la taille de l'édifice à construire. $H$ est la hauteur, $l$ la largeur et $L$ la longueur |
| *nb_type* | Nombre de types de modules. Chaque type de module correspond à une fonction particulière dans l'édifice |
| *nb_mode* | Nombre de modes possibles pour chacun des modules |
| *duree_type_mode* | Matrice de taille *[nb_mode ; nb_type]* qui informe sur la durée d'installation d'un type de module suivant les différents modes de construction |
| *cout_type_mode* | Matrice de taille *[nb_mode ; nb_type]* qui informe sur la durée d'installation d'un type de module suivant les différents modes de construction |

L'objectif est donc de construire une liste des lots de travail avec leurs caractéristiques représentant une instance de projet. Celle-ci sert pour la génération de l'état de planification initial dans le modèle.

Tout d'abord, on commence par initialiser une matrice vide dont les colonnes sont présentées dans le Tableau 5.3. On utilise en effet plusieurs tableaux de références afin de configurer l'algorithme. Ces tableaux permettent d'assurer que l'ensemble des instances générées aient la même structure, c'est un élément incontournable afin que le modèle puisse apprendre adéquatement.

Nous utilisons alors 3 tableaux de références :

- Zone supplémentaire : ce tableau indique si le lot de travail nécessite une grue pour l'installer. Ce sont les modes de construction modulaire qui ont généralement besoin d'une

grue. Pour chaque type de module, et chaque mode de construction, le tableau présente un 1 si une grue est nécessaire, un 0 sinon ;

- Durée : ce tableau indique la durée du lot de travail pour chaque type de module et chaque mode de construction ; et
- Coût : ce tableau indique le coût du lot de travail pour chaque type de module et chaque mode de construction.

Ces tableaux ont été remplis avec des valeurs indicatives à la suite de recherches sur les pratiques de construction classiques et modulaire (Bertram, Nick et al., 2019 ; Lopez, Diana et al., 2016).

Ensuite, l'algorithme choisit pour chaque module un emplacement et un type aléatoirement. À partir du module, on génère alors autant de lots de travail que de mode de construction. On ajoute ensuite la durée, le coût du lot de travail et le besoin de zone supplémentaire à partir des tableaux de référence.

Enfin on renseigne les colonnes de précédence et d'exclusion. Pour les précédences, on repère les lots de travail qui doivent être effectués en dessous du lot de travail considéré et les on ajoute sous forme de liste dans la colonne de précédence. En effet, il est physiquement impossible d'installer un module sans que le module sous celui-ci ait été préalablement installé.

Pour la colonne exclusion, on repère les lots de travaux qui concernent un même module, mais avec un mode de construction différent. On les ajoute ensuite sous forme de liste dans la colonne. Si un module est planifié avec un certain mode de construction, tous les autres lots de travaux concernant le même module ne pourront plus être planifiés, ce qui est logique.

Ainsi, on obtient une liste de travaux avec leurs caractéristiques qui est utilisée dans le modèle comme initialisation.

Tableau 5.3 Définition des colonnes pour la génération des instances

| Colonne | Définition | Source |
|---|---|---|
| ID | Nombre généré automatiquement permettant d'identifier le lot de travail | Automatique |
| Type | Type du module | Aléatoire parmi les types |
| Mode | Mode de construction | Déterminé grâce aux tableaux de référence sur les modes de construction |
| X | 3 entiers représentant les 3 coordonnés du module dans l'espace | Aléatoire dans l'intervalle [0,l] |
| Y | | Aléatoire dans l'intervalle [0,L] |
| Z | | Aléatoire dans l'intervalle [0,H] |
| Précédence | Liste des lots de travail qui doivent être terminé avant de planifier le lot de travail | Déterminé dans l'algorithme à partir de la définition des emplacements des autres lots de travail |
| Exclusion | Liste des lots de travail non compatible avec le lot de travail | Déterminé dans l'algorithme à partir des autres lots de travail |
| Zone supplémentaire | Nombre d'espace supplémentaire nécessaire afin de planifier le lot de travail. Ceci correspond à l'utilisation de la grue dans certains modes de construction | Déterminé à partir du tableau de référence sur les zones supplémentaires |
| Durée | Durée du lot de travail | Déterminé à partir du tableau de référence sur les durées |

Tableau 5.3 Définition des colonnes pour la génération des instances (suite et fin)

| | | |
|---|---|---|
| Coût | Coût du lot de travail | Déterminé à partir du tableau de référence sur les coûts |

## 5.2 Expériences

### 5.2.1 Expérience 1 : Validation de l'apprentissage

Cette expérimentation a pour objectif de montrer que le modèle est bien capable d'apprendre. Afin de savoir si le modèle apprend, on peut observer l'évolution du score obtenu par le modèle. En effet, l'objectif du modèle est d'apprendre de façon à obtenir un échéancier de plus en plus performant. Pour ce faire, nous avons choisi des valeurs pour les paramètres de modèle présentées dans le Tableau 5.5. Ces valeurs ont été déterminées à la suite d'une succession de tests sur les différents paramètres.

L'évolution du score est alors présentée dans la Figure 5.1. On a aussi ajouté une barre à chaque fois que le réseau de neurones est mis à jour. Ainsi on peut noter plusieurs éléments. Tout d'abord, malgré un score qui augmente dans les premières itérations, le score a une tendance à diminuer jusqu'à se stabiliser. On peut voir dans les expérimentations suivantes qu'à partir d'un certain nombre d'itérations, le modèle converge systématiquement vers un score.

Ensuite, on peut remarquer qu'après chaque mise à jour du réseau de neurones, le score diminue. C'est très intéressant, car cela montre qu'à chaque mise à jour, le réseau de neurones est plus performant. On peut alors valider l'apprentissage du modèle.

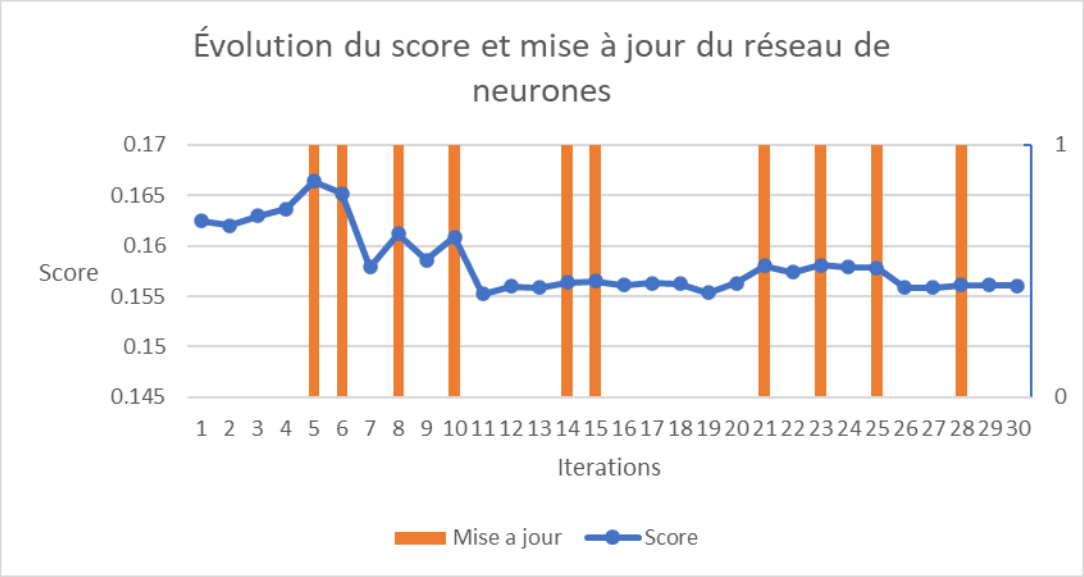


Figure 5.1 Critères de mise à jour du modèle

### 5.2.2 Expérience 2 : Variation des paramètres

#### 5.2.2.1 **Protocole et objectifs**

Cette section présente le protocole qui a été suivi lors de cette expérimentation dont l'objectif est d'observer la variation des performances de l'algorithme en fonction de la variation de certains paramètres présentés plus haut. Pour chacun des paramètres, des valeurs ont été déterminées à l'intérieur de leur intervalle de définition afin d'effectuer les tests. Lors de chaque expérimentation, les autres paramètres restent constants afin d'observer seulement l'influence du paramètre testé. Leurs valeurs ont été déterminées grâce à des tests préliminaires permettant d'assurer l'apprentissage de l'algorithme dans un temps raisonnable. Ainsi, les valeurs de référence de chacun des paramètres sont présentées dans le Tableau 5.5.

Nous avons choisi d'expérimenter seulement certains paramètres selon leur intérêt a priori. Ainsi, nous avons porté nos tests sur les paramètres suivants :

- Taille ;
- numIters ;
- numEps ;
- updateThreshold ; et

- numMCTSSims.

Tableau 5.5 Valeurs références des paramètres du modèle

| Paramètres | Valeur de référence |
|---|---|
| $Taille = [H, l, L]$ | *[2,3,3]* |
| *numIters* | *30* |
| *numEps* | *100* |
| *updateThreshold* | *0.6* |
| *numMCTSSims* | *40* |
| *MaxlenofQueue* | *200000* |
| *arenaCompare* | *20* |
| *F* | $\sum a.d_i + c_i$ (11)<br>Où $d_i$ et $c_i$ sont les durées et les coûts des lots de travail *i* planifié dans l'échéancier |
| *a*<br>Constante représentant le coût d'une journée de travail | *800* |

L'algorithme suit une logique itérative pour laquelle, à chaque itération, on entraine un réseau de neurones dont on compare les performances à celles de sa meilleure version. Cette comparaison est réalisée *arenaCompare* fois afin de pouvoir comparer en moyenne. On observera alors pour chaque itération l'évolution du score en moyenne. Ce score est présenté sous forme de pourcentage

afin de faciliter les comparaisons. Ce pourcentage est calculé suivant la formule ci-dessous, comme le résultat de la fonction d'évaluation *F* divisé par sa valeur maximum calculée en considérant les durées et les coûts de tous les lots de travail.

$$score = \frac{F(s_f)}{\max(F)} \tag{12}$$

Nous regarderons également le nombre de fois où les poids du réseau de neurones ont été mis à jour. Cet indicateur est intéressant, car il nous informe sur l'évolution de l'apprentissage. Il est à croiser avec l'évolution du score afin de comprendre l'impact de chaque mise à jour des poids du réseau de neurones sur les performances.

Un des enjeux majeurs de l'apprentissage automatique concerne le temps de calcul. En effet, un modèle d'apprentissage automatique est intéressant si le temps de calcul reste raisonnable dans le cadre de son application. Dans le cadre de ce travail exploratoire, l'objectif n'étant pas d'optimiser le modèle, mais d'en mesurer l'intérêt, la mesure du temps de calcul reste indicative. Nous regarderons ainsi l'évolution du temps de calcul suivant les différents paramètres afin de valider l'intérêt du modèle.

#### 5.2.2.2 **Variation de *numIters***

*numIters* correspond au nombre d'itérations du modèle. Chaque itération présente une opportunité pour le modèle de mettre à jour les poids du réseau de neurones et donc d'améliorer l'évaluation des probabilités et de la valeur probable de l'état de planification. Le Tableau 5.6 présente l'intervalle de définition de *numIters* ainsi que les valeurs choisies pour les tests.

Tableau 5.6 Définition des tests pour numIters

| Paramètre | *numIters* |
|---|---|
| Intervalle de définition | $[\![1; \infty]\!]$ |
| Référence | *30* |
| Test 1 | *10* |
| Test 2 | *50* |

#### *5.2.2.2.1 Évolution du score*

Dans la Figure 5.2, on observe alors l'évolution du score en fonction du nombre d'itérations. On remarque que le modèle semble converger dès l'itération 12 environ pour le test avec 30 itérations, alors que le score semble se stabiliser un peu plus tôt vers l'itération 8 pour le test avec 50 itérations. En revanche, le test avec 10 itérations ne semble pas encore converger. Ceci s'explique, car le modèle a besoin d'un certain nombre d'itérations avant de converger vers le score optimal. On voit également dans le Tableau 5.7 que le nombre de mises à jour du réseau de neurones augmente avec le nombre d'itérations. Le tableau montre que les mises à jour sont plus fréquentes dans les premières itérations. On peut l'expliquer, car au départ, lors des premières itérations, le réseau de neurones est encore peu entrainé et fournit alors des résultats peu performants. Le modèle est globalement moins performant et il est alors plus facile de trouver une meilleure configuration des poids du réseau de neurones. Petit à petit, le modèle devient plus performant ; il est donc plus difficile de l'améliorer. Il y a donc moins de mises à jour lors des dernières itérations.

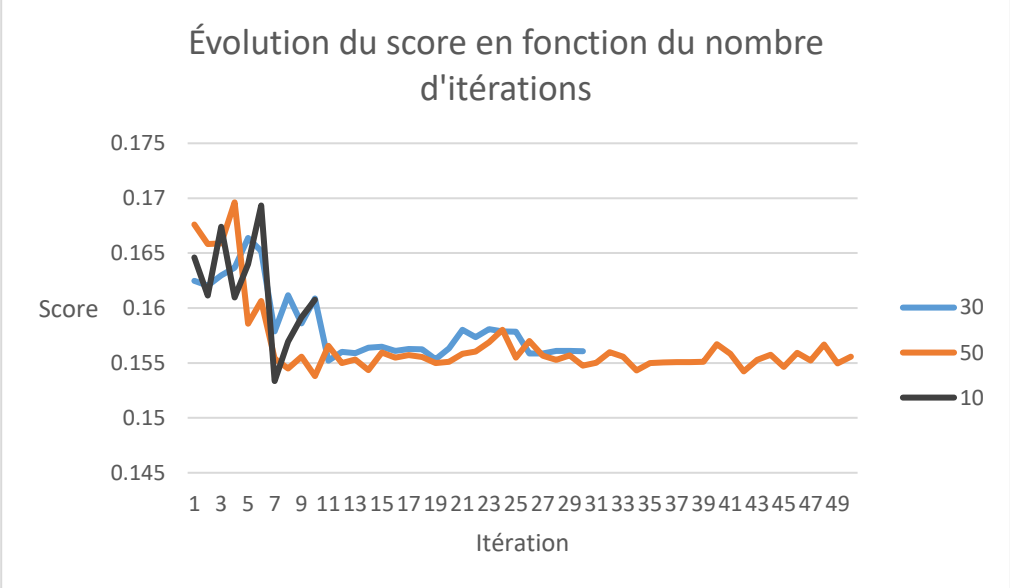


Figure 5.2 Évolution du score en fonction du nombre d'itérations

Tableau 5.7 Évolution du nombre de mises à jour du réseau de neurones en fonction du nombre d'itérations

| *numIters* | Nombre de mises à jour du réseau de neurones | Proportion d'itérations avec une mise à jour du réseau de neurones |
|---|---|---|
| 10 | 6 | 60% |
| 30 (référence) | 12 | 40% |
| 50 | 14 | 28% |

*5.2.2.2.2 Évolution du temps de calcul*

Dans le Tableau 5.8, on observe la variation du temps de calcul en fonction du nombre d'itérations. Le temps de calcul augmente avec le nombre d'itérations. Ceci s'explique assez naturellement, car les opérations pour chaque itération sont très similaires et concernent une quantité de données du même ordre de grandeur. En effet, comme précisé plus tôt, l'ensemble d'entrainement est alimenté à chaque itération jusqu'à un certain seuil fixé. Une fois ce seuil atteint, les plus vieux exemples sont supprimés de l'ensemble d'entrainement ; l'ensemble d'entrainement garde alors constamment la même taille. Une autre analyse plus poussée de l'évolution du temps de calcul pourrait permettre de mesurer l'impact de l'entrainement du réseau de neurones en comparaison avec le temps de calcul des autres opérations algorithmiques, notamment celles de l'arbre de recherche de Monte-Carlo.

Tableau 5.8 Évolution du temps de calcul en fonction du nombre d'itérations

| Test | *numIters* | Temps de calcul (en secondes) | Temps de calcul (en jours) | Écart relatif |
|---|---|---|---|---|
| Test 1 | *10* | *48816* | *0.565* | *-69%* |
| Référence | *30* | *157474* | *1.823* | *NA* |
| Test 2 | *50* | *443288* | *5.130* | *+181%* |

*5.2.2.2.3 Intérêt du nombre d'itérations*

En conclusion, on peut affirmer que le nombre d'itérations est un paramètre qui influe beaucoup sur la performance finale de l'algorithme. Par conséquent, il est nécessaire de trouver le seuil permettant au modèle de converger vers son score optimal. Ce paramètre est aussi très impactant sur le temps de calcul. Le nombre de références de 30 itérations a été choisi afin de s'assurer que le modèle converge systématiquement, quelles que soient les valeurs des paramètres.

5.2.2.3 **Variation de *numEps***

*numEps* correspond au nombre de simulations qui sont effectuées au début de chaque itérations. Pour chaque simulation, tous les états de planification correspondants sont ajoutés à l'ensemble d'entrainement qui permet d'entrainer le réseau de neurones. A chaque fois, on rajoute de nouvelles données d'entrainement, générées avec un modèle de plus en plus performant. C'est le principe de l'apprentissage par renforcement.

Cette expérimentation permet alors d'évaluer l'influence de la variation de *numEps* sur les performances du modèle. Les valeurs choisies afin de réaliser les tests, ainsi que l'intervalle de définition de ce paramètre, sont présentés dans le Tableau 5.9.

Tableau 5.9 Définition des tests pour *numEps*

| Paramètre | *numEps* |
|---|---|
| Intervalle de définition | $[\![1; \infty]\!]$ |
| Référence | *100* |
| Test 1 | *20* |
| Test 2 | *200* |
| Test 3 | *500* |

#### *5.2.2.3.1 Évolution du score*

La Figure 5.3 représente l'évolution du score pour 30 itérations en modifiant le paramètre *numEps* avec les valeurs 20, 100, 200 et 500. Tout d'abord, on peut remarquer que lorsque *numEps* vaut 20, le modèle a une courbe de progression beaucoup plus lente que pour les autres valeurs. On peut affirmer qu'après 30 itérations, le modèle commence à peine à converger. On comprend alors que pour des valeurs faibles, le modèle tarde à converger. De plus, sa performance est moins bonne à la fin des 30 itérations.

On remarque également que le modèle semble converger plus vite plus *numEps* est grand. En effet, le modèle converge vers l'itération 5 avec la valeur 500, vers l'itération 8 pour la valeur 200 et vers l'itération 13 pour la valeur 100 qui est la référence. Ceci peut s'expliquer car plus le nombre de simulations est grand, plus l'ensemble d'entrainement est grand au départ, ce qui permet au réseau de neurones d'apprendre plus vite. La politique de choix du réseau de neurones qui se définit grâce à l'apprentissage par renforcement est alors bien déterminée dès les premières itérations, ce qui lui permet de converger vers un résultat plus rapidement.

Cependant, les états de planification des premières itérations sont générés à partir de versions du réseau de neurones peu performantes. Ainsi, le modèle risque de converger vers un optimum local, comme c'est le cas avec 500 simulations. On peut alors soupçonner un surapprentissage du réseau de neurones. Le surapprentissage est un biais courant en apprentissage automatique. Quand il y a

surapprentissage, le modèle a appris en étant trop proche de données d'entrées particulières. Ainsi, il n'est plus capable de proposer des résultats intéressants pour des données légèrement différentes.

En effet, en se basant majoritairement sur des données d'entrainement générées par un modèle peu performant, le réseau de neurones est largement influencé par ces premières phases d'entrainement alors que c'est précisément dans les premières itérations que le modèle évolue le plus. Il stagne alors rapidement à un niveau de performance relativement faible. Qui plus est, ceci l'empêche par la suite de générer des états de planification qui lui permettraient de proposer des échéanciers plus intéressants.

On comprend avec les courbes pour 100 simulations et 200 simulations qu'un équilibre doit être trouvé afin de pouvoir converger assez rapidement grâce à un nombre de simulations suffisamment grand pour générer assez de données d'entrainement, mais suffisamment petit afin d'éviter le surapprentissage. Dans notre cas, c'est donc avec 200 simulations que le modèle est le plus performant.

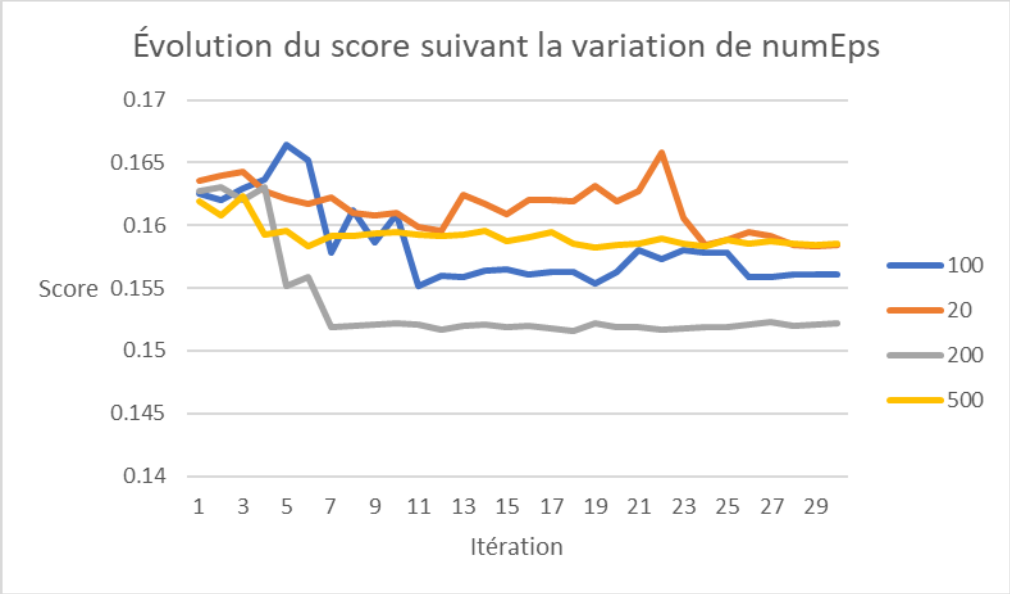


Figure 5.3 Évolution du score en fonction de la variation de *numEps*

#### *5.2.2.3.2 Évolution du temps de calcul*

Le Tableau 5.10 rassemble les temps de calcul pour les différents tests effectués en faisant varier les valeurs de nombre de simulations. On remarque que le temps de calcul évolue significativement en fonction de ce paramètre. En effet, on constate une diminution de 58% pour 20 simulations par rapport au modèle de référence et une augmentation respectivement de 90% et 311% pour 200 et 500 simulations. La relation entre le nombre de simulations et le temps de calcul s'explique naturellement, car à chaque itération, on commence par générer des états de planification pour entrainer le modèle grâce à ces simulations. Donc, plus on opère de simulations, plus le temps de calcul est long.

Tableau 5.10 Évolution du temps de calcul selon la variation de *numEps*

| Test | *numEps* | Temps de calcul (en secondes) | Temps de calcul (en jours) | Écart relatif |
|---|---|---|---|---|
| Test 1 | *20* | *66195* | *0.766* | *-58%* |
| Référence | *100* | *157474* | *1.823* | *NA* |
| Test 2 | *200* | *298958* | *3.460* | *90%* |
| Test 3 | *500* | *646840* | *7.487* | *311%* |

#### *5.2.2.3.3 Intérêt du nombre de simulations*

On peut conclure que le nombre de simulations est déterminant pour le modèle. En effet, pour obtenir les meilleures performances, il est nécessaire de trouver un bon équilibre avec un nombre de simulations optimal. De plus, il est très impactant sur le temps de calcul, donc c'est un paramètre important quand l'objectif est d'optimiser le modèle. La valeur référence de 100 simulations a été choisie afin de pouvoir effectuer un maximum d'expérimentations dans un temps raisonnable.

#### 5.2.2.4 **Variation du seuil de mise à jour des poids du réseau de neurones**

Le seuil de mise à jour des poids correspond au pourcentage minimum de victoire nécessaire au réseau de neurones temporaire pour être considéré comme meilleur que le réseau de neurones optimal. Cette comparaison est effectuée à la fin de chaque itération. Pour ce paramètre, nous avons choisi 4 valeurs présentées dans le Tableau 5.11.

Tableau 5.11 Définition des tests pour le seuil de mise à jour

| Paramètre | Seuil de mise à jour |
|---|---|
| Intervalle de définition | [0,5 ;1] |
| Référence | *0,6* |
| Test 1 | *0,5* |
| Test 2 | *0,8* |
| Test 3 | *0,9* |

##### *5.2.2.4.1 Évolution du score*

La Figure 5.4 montre l'évolution du score au cours des itérations du modèle en fonction de plusieurs valeurs du seuil de mise à jour. On peut remarquer que quand le seuil vaut 0.5, le modèle semble converger plus vite dès l'itération 8, pourtant il semble également moins stable à partir de l'itération 18. En effet, cela peut s'expliquer, car si le seuil est bas, les poids du réseau de neurones sont mis à jour plus souvent, même dans le cas où le réseau de neurones temporaire est à peine meilleur que le réseau de neurones optimal. Ces changements récurrents n'assurent pas que la nouvelle version du réseau de neurones soit significativement meilleure. Dans le même sens, on remarque quand le seuil est plus haut, par exemple pour 0.8 ou 0.9, que le modèle est plus lent à converger. Ceci s'explique par le fait que le réseau de neurones temporaire doit être largement plus

performant que le réseau de neurones optimal pour être mis à jour. Les changements sont alors moins récurrents, mais chaque changement assure une bonne amélioration du modèle.

Malgré ces explications, la performance du modèle reste très similaire, même avec des valeurs de seuil différentes. L'impact de ce paramètre est relativement faible. On peut juste souligner qu'un seuil trop faible pourrait rendre le modèle moins stable.

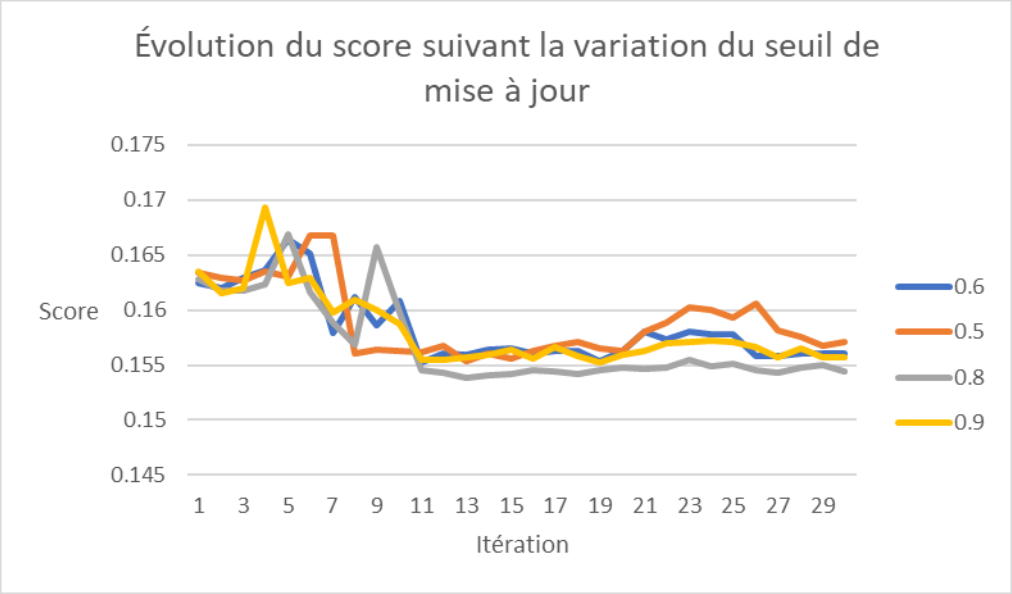


Figure 5.4 Évolution du score suivant la variation du seuil de mise à jour

*5.2.2.4.2 Évolution du temps de calcul*

Les temps de calcul suivant les différentes valeurs du seuil de mise à jour sont présentés dans le Tableau 5.12. On peut voir que le temps de calcul ne varie très peu. Ceci n'est pas surprenant, car quel que soit le seuil, les opérations effectuées restent majoritairement les mêmes.

Tableau 5.12 Évolution du temps de calcul selon la variation du seuil de mise à jour

| Test | *updateThreshold* | Temps de calcul (en secondes) | Temps de calcul (en jours) | Écart relatif |
|---|---|---|---|---|
| Test 1 | *0.5* | *156683* | *1.813* | *-1%* |
| Référence | *0.6* | *157474* | *1.823* | *NA* |
| Test 2 | *0.8* | *164052* | *1.899* | *+4%* |
| Test 3 | *0.9* | *169766* | *1.965* | *+8%* |

*5.2.2.4.3 Intérêt du seuil de mise à jour*

Le seuil de mise à jour est finalement un paramètre relativement peu impactant sur les performances du modèle. On a vu qu'il influe sur la stabilité du modèle ainsi que sur la vitesse de convergence, mais très peu sur les performances finales.

5.2.2.5 **Variation de *numMCTSSims***

Le nombre de simulations du MCTS correspond au nombre d'itérations qu'effectue l'arbre de recherche de Monte-Carlo avant de fournir les probabilités permettant de faire le choix du module à planifier. Ainsi, on peut penser que plus l'arbre de recherche fait des simulations, plus les probabilités seront bonnes, car l'arbre de recherche a été visité plus de fois. Pour ce paramètre, nous avons décidé de choisir 4 valeurs présentées dans le Tableau 5.13.

Tableau 5.13 Définition des tests pour le nombre de simulations du MCTS

| Paramètre | *numMCTSSims* |
|---|---|
| Intervalle de définition | $[\![1; \infty]\!]$ |
| Référence | *40* |
| Test 1 | *5* |
| Test 2 | *10* |
| Test 3 | *80* |

#### *5.2.2.5.1 Évolution du score*

La Figure 5.5 montre l'évolution du score au cours des itérations du modèle en fonction de la variation du nombre de simulations de l'arbre de recherche. On remarque que plus le nombre de simulations est grand, plus le modèle est performant.

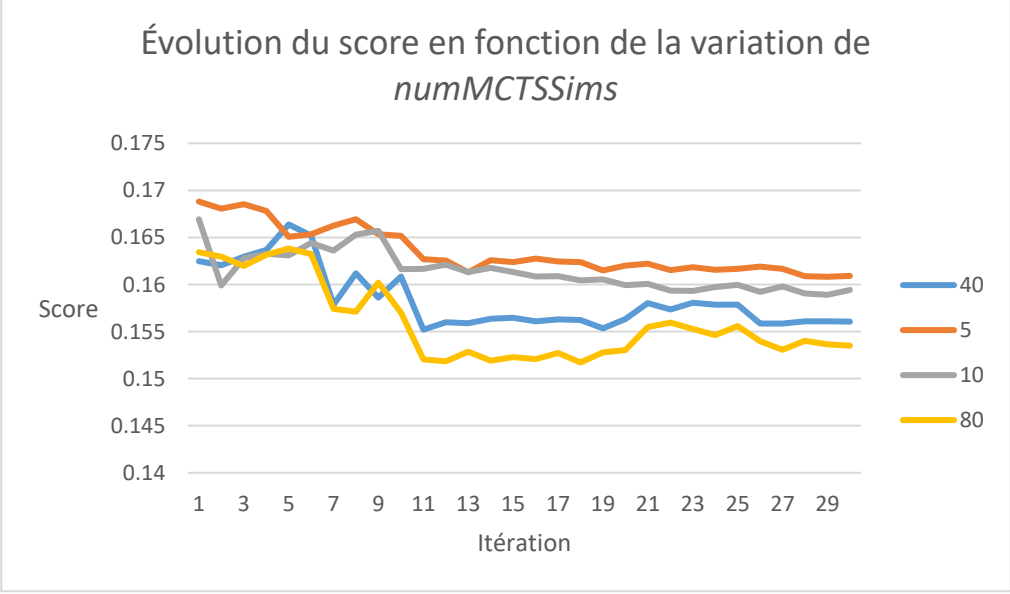


Figure 5.5 Évolution du score en fonction de la variation de *numMCTSSims*

*5.2.2.5.2 Évolution du temps de calcul*

Les temps de calcul suivant les différentes valeurs de *numMCTSSims* sont présentés dans le Tableau 5.12. On peut voir que le temps de calcul augmente considérablement lorsque le nombre de simulations augmente.

Tableau 5.14 Évolution du temps de calcul selon la variation de *numMCTSSims*

| Test | *numMCTSSims* | Temps de calcul (en secondes) | Temps de calcul (en jours) | Écart relatif |
|---|---|---|---|---|
| Test 1 | *5* | *16373* | *0.189502315* | *-90%* |
| Test 2 | *10* | *29214* | *0.338125* | *-80%* |
| Référence | *40* | *157474* | *1.823* | *NA* |
| Test 3 | *80* | *450498* | *5.214097222* | *+186%* |

*5.2.2.5.3 Intérêt du nombre de simulations de l'arbre de recherche*

Le nombre de simulations de l'arbre de recherche est un paramètre déterminant dans les performances du modèle. On voit une nette amélioration quand le nombre de simulations est de 80. Cependant, le temps de calcul augmente également. Dans la pratique, il faudrait alors trouver un équilibre de façon à obtenir une bonne performance dans des temps raisonnables. C'est pour cette raison que nous avons choisi 40 simulations en tant que valeur de référence.

5.2.2.6 **Variation de la taille de l'édifice**

La taille de l'édifice à construire est un paramètre intéressant à analyser, car il permet de mesurer l'applicabilité du modèle pour des projets de taille différente.

Le nombre de modules est directement calculé en fonction du nombre de cubes dans l'espace, étant donné que l'on planifie un module par cube. De plus, nous avons considéré 3 modes de construction

dans les tests. Pour chaque module, on a alors 3 lots de travail. Ces informations sont résumées dans le Tableau 5.15.

Tableau 5.15 Définition des tests pour la taille de l'édifice

| Paramètre | *Taille* | Nombre de modules | Nombre de lots de travail |
|---|---|---|---|
| Référence | *[2,3,3]* | *18* | *54* |
| Test 1 | *[2,4,4]* | *32* | *96* |
| Test 2 | *[2,5,5]* | *50* | *150* |
| Test 3 | *[3,3,3]* | *27* | *81* |

#### *5.2.2.6.1 Évolution du score*

La Figure 5.6 montre l'évolution du score au cours des itérations du modèle en fonction de la variation de la taille des projets. Malgré quelques exceptions, il semblerait que la taille du projet n'influe pas sur la performance du résultat.

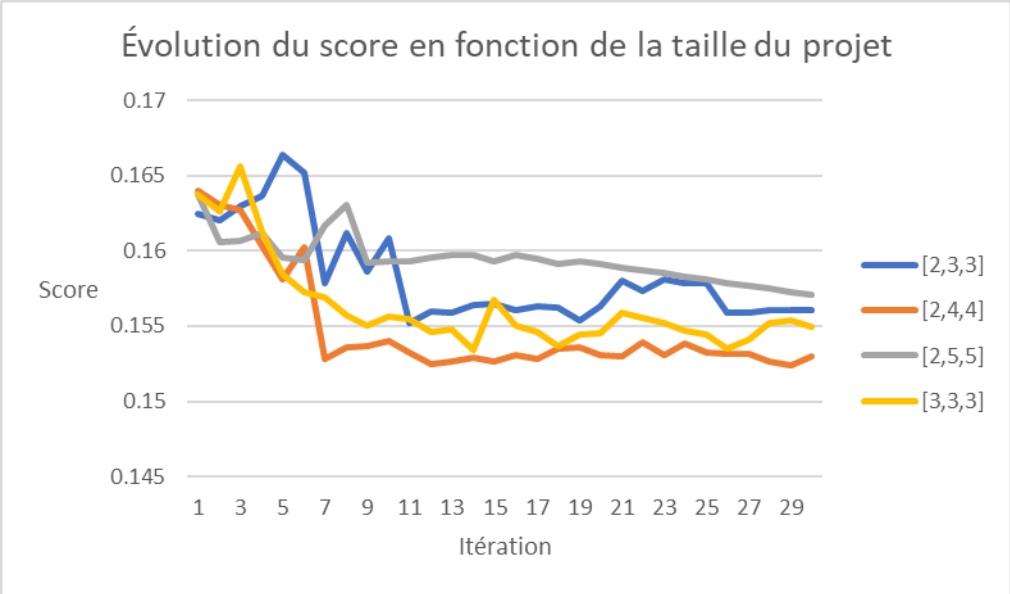


Figure 5.6 Évolution du score en fonction de la taille du projet

*5.2.2.6.2 Évolution du temps de calcul*

Les temps de calcul suivant les différentes tailles des projets sont présentés dans le Tableau 5.16. On peut voir que le temps de calcul augmente considérablement avec le nombre de lots de travail. C'est une observation habituelle lors de la résolution de problèmes de planification.

Tableau 5.16 Évolution du temps de calcul selon la taille des projets

| Test | *Taille* | Nombre de lots de travail | Temps de calcul (en secondes) | Temps de calcul (en jours) | Écart relatif |
|---|---|---|---|---|---|
| Référence | *[2,3,3]* | *54* | *157474* | *1.823* | *NA* |
| Test 1 | *[2,4,4]* | *96* | *405547* | *4.694* | *+158%* |
| Test 2 | *[2,5,5]* | *150* | *572715* | *6.628645833* | *+264%* |
| Test 3 | *[3,3,3]* | *81* | *447588* | *5.180416667* | *+184%* |

*5.2.2.6.3 Intérêt de la taille des projets*

Le paramètre de la taille des projets influe peu sur les performances du modèle. On peut penser que même pour de grands projets, les performances seront similaires. Cependant, comme pour beaucoup de méthode de résolution de problèmes de planification, le temps de calcul augmente considérablement en fonction du nombre de lots de travail. On en déduit que la planification de grands projets de construction avec beaucoup plus de lots de travail demandera un temps d'entrainement très important, qui peut consister une limite du modèle.

## 5.2.3 Expérience 3 : Comparaison avec les autres modèles

5.2.3.1 **Protocole et objectif**

Maintenant que l'apprentissage du modèle est validé et que l'analyse de l'influence des paramètres sur les performances du modèle est faite, il peut être intéressant de comparer ce modèle avec des modèles très simples.

Pour ce faire, deux autres modèles ont été choisis :

- Choix aléatoire ; et
- Mode de construction classique.

Les deux modèles ont été développés afin de proposer des échéanciers à partir des mêmes instances de projet, et avec la même logique. En effet, pour les deux modèles, on choisit un lot de travail à planifier tour après tour. Mais alors que notre modèle utilise l'arbre de recherche ainsi que le réseau de neurones pour guider le choix, les autres modèles choisissent le lot de travail à planifier directement parmi les lots de travail restant à planifier.

Le modèle de choix aléatoire choisit aléatoirement un lot de travail à planifier parmi les lots de travail planifiables.

Le modèle de mode de construction classique choisit uniquement les lots de travail qui utilisent le mode de construction classique. Parmi ceux disponibles, le choix est porté sur le lot de travail avec la durée la plus courte. Cette règle de priorité est classiquement utilisée dans les méthodes de résolution du modèle CPM.

Afin de comparer ces modèles avec notre modèle, nous entrainons notre modèle avec les paramètres qui se sont révélés être les plus performants dans nos précédents tests ; ceux-ci sont présentés dans le Tableau 5.17.

Ensuite, nous générons 100 instances de projets sur lesquels les 3 modèles proposent un échéancier. On compare alors en moyenne les résultats du modèle aléatoire avec notre modèle, puis les résultats du modèle du mode de construction classique. Ce résultat est présenté sous forme du pourcentage de fois où notre modèle est meilleur que le modèle considéré sur les 100 instances.

Tableau 5.17 Paramètres pour la comparaison avec les autres modèles

| Paramètres | Valeur de référence |
|---|---|
| $Taille = [H, l, L]$ | *[2,3,3]* |
| *numIters* | *30* |
| *numEps* | *200* |
| *updateThreshold* | *0.6* |
| *numMCTSSims* | *80* |
| *MaxlenofQueue* | *200000* |
| *arenaCompare* | *200* |
| *F* | $\sum a.d_i + c_i$ (13)<br>Où $d_i$ et $c_i$ sont les durées et les coûts des lots de travail $i$ planifiés dans l'échéancier |
| *a*<br>Constante représentant le coût d'une journée de travail | *800* |

5.2.3.2 **Résultats et analyse**

Les résultats sont présentés dans le Tableau 5.18. On peut remarquer que le modèle développé dans l'étude est plus performant que le modèle de choix aléatoire pour 93 % des instances. On peut alors conclure que notre modèle est meilleur qu'un choix aléatoire.

Cependant, notre modèle n'est meilleur que le modèle du mode construction classique que pour 5 % des instances de projet. On pourrait alors conclure que notre modèle est moins performant.

Pourtant, il est à noter dans notre modèle, les lots de travaux utilisant un mode de construction classique n'ont pas le besoin d'utiliser une grue. Ainsi, un lot de travail n'occupe qu'un seul espace dans notre modèle pour ces lots de travail. Au contraire, des lots de travail utilisant une grue utilisent un deuxième espace le temps de l'installation du module. On peut alors supposer que cette caractéristique est très impactante pour les projets avec des petits espaces, puis l'impact diminue avec les grands projets. On peut alors vérifier cette hypothèse en proposant d'effectuer la même expérimentation avec un espace au sol plus grand. Le résultat est présenté dans le Tableau 5.18 avec des projets de taille [2, 5, 5]. On a alors 25 cubes par niveau contre 9 dans l'expérimentation précédente. On voit alors une légère amélioration sur les performances de notre modèle. Cependant, cela peut constituer une limite de notre modèle.

De plus, on comprend que la comparaison avec le modèle du mode de construction classique est très dépendante des tableaux de référence sur les coûts et les durées qui servent d'entrée à notre modèle. En effet, en plus de la limitation présentée précédemment, il parait logique que si les durées et les coûts des lots de travail pour le mode de construction classique sont bien plus avantageux que ceux des autres modes, le modèle ne pourra pas proposer de meilleurs échéanciers en choisissant les autres modes. Les tableaux de références de coûts et des durées ont pourtant été déterminés en prenant en compte la réalité de la construction. On pourrait tout de même vérifier cette hypothèse en effectuant la même expérimentation, mais avec des durées et des coûts qui avantagent plus les modes de construction non classiques. Le résultat est présenté dans le Tableau 5.18. On voit une nette amélioration avec un pourcentage de victoire de 55 %. Ceci montre bien que notre modèle est capable d'apprendre et d'être plus performant qu'un modèle qui ne choisirait qu'un seul mode de construction.

Mais on comprend également que le modèle est très dépendant des tableaux de référence ainsi que de la modélisation de l'utilisation de la grue. Ces éléments constituent les principales limitations de notre modèle.

Tableau 5.18 Résultats des comparaisons avec les autres modèles

| | Modèle de choix aléatoire | Modèle du mode de construction classique |
|---|---|---|
| Pourcentage de victoire du modèle de l'étude | 93% | 5% |
| Pourcentage de victoire du modèle de l'étude avec des projets de taille [2,5,5] | 94% | 9% |
| Pourcentage de victoire du modèle de l'étude avec des coûts et durée revus | 97% | 55% |

## 5.3 Conclusion

Dans ce chapitre, les étapes d'expérimentation et de validation de la méthode de recherche ont été développées. Nous avons montré que notre modèle est bien capable de progresser afin de proposer des échéanciers de projets de plus en plus performants. L'analyse de sensibilité des paramètres a permis de voir que certains paramètres comme le nombre de simulations du MCTS ainsi que le nombre d'exemples générés à l'initialisation de chaque itération ont des impacts importants sur la performance du modèle. De plus, on peut noter que la taille du projet n'influe pas, mais que des projets de taille importante demandent un temps de calcul très long qui peut constituer une limite de notre modèle. Enfin, on a montré que notre modèle est plus performant que le choix aléatoire d'un lot de travail, mais qu'il connait des difficultés contre le modèle du mode de construction classique avec la règle de priorité sur les lots de travail le plus court. Ceci constitue également une des limites du modèle qui sont présentées dans le chapitre 6.

# CHAPITRE 6 CONCLUSION ET RECOMMANDATIONS

La revue de littérature a permis de montrer l'inexistence de modèle permettant de générer des échéanciers pour la planification tactique dans les premières phases du projet, en prenant en compte les différents modes de construction afin de les intégrer dans l'échéancier, plutôt que de faire le choix du mode construction dès le départ du projet. De plus, le travail de Smith (2018) a retenu notre attention en proposant un modèle de séquencement afin d'installer des pièces de construction en utilisant la combinaison d'un arbre de recherche de Monte-Carlo et d'un réseau de neurones. Ainsi, nous avons décidé de développer un modèle dans ce contexte en utilisant un modèle similaire.

Pour mener à bien ce projet, nous avons suivi une méthode de recherche qui consiste à définir le cadre dans lequel le modèle a été développé en faisant une analogie avec la théorie de jeux. Nous avons alors proposé un modèle que nous avons expérimenté et validé grâce à des instances de projets générées pour ce projet de recherche. Cependant, nous n'avons pas pu vérifier l'applicabilité de l'outil dans le contexte réel dû au manque de données réelles dans ce contexte. Ainsi, nous avons rempli la majorité de nos objectifs de recherche.

D'un point de vue théorique, ce travail permet de proposer un nouveau modèle de planification initiale, qui a la caractéristique supplémentaire de prendre en compte les modes de constructions alternatifs. Ces modes ne sont encore que peu présents dans la littérature de recherche dans le cadre de la planification initiale, car dans l'industrie de la construction, le choix du mode de construction est bien souvent guidé par l'évaluation primaire du coût total du projet, mais également par l'expérience de l'entreprise dans ces autres modes de construction. Ce choix n'est que très rarement fait en fonction des enjeux de planification. Notre modèle propose une première possibilité dans ce sens. De plus, nous avons proposé une analogie avec la théorie des jeux afin de définir le cadre de notre modèle qui a permis d'utiliser l'apprentissage automatique en adaptant des modèles similaires.

Notre étude comporte également des limites que nous avons constatées dans ce mémoire. En effet, la première limite concerne l'applicabilité de ce modèle dans un contexte réel. Ce modèle demande beaucoup de temps de calcul pour les projets de grande envergure, alors qu'il aurait un intérêt particulier pour des projets comprenant beaucoup de lots de travaux, dont la proposition d'échéancier est difficilement réalisable manuellement.

Des perspectives de recherche découlent également de ce mémoire. En effet, ce travail exploratoire est concentré sur la découverte, mais ne propose pas un réseau de neurones optimal. Il pourrait alors être intéressant de revoir la modélisation du réseau de neurones et d'observer l'amélioration des performances du modèle. De plus, on a remarqué que le modèle dépend beaucoup des tableaux de référence, déterminés dans notre cas approximativement selon les pratiques de l'industrie observées. Il pourrait être intéressant d'adapter le modèle afin de permettre la détermination des coûts et des durées des lots de travail de manière plus précise et directement intégrée.

En conclusion, ce travail exploratoire propose un premier modèle utilisant l'apprentissage automatique qui définit des échéanciers pour les projets de construction dans le cadre d'une planification initiale en considérant les modes de constructions alternatifs comme la construction modulaire, qui présente de plus en plus d'intérêt dans l'industrie de la construction.

# RÉFÉRENCES

AACE International. (2010). ACE International recommended practice No. 27R-03 Schedule classification system. :

Alcaraz, J., Maroto, C., & Ruiz, R. (2003). Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of the Operational Research Society, 54(6), 614-626.

Alpaydin, E. (2020). Introduction to machine learning : MIT press.

Bertram, N., Fuchs, S., Mischke, J., Palter, R., Strube, G., & Woetzel, J. (2019). Modular construction: From projects to products. McKinsey & Company: Capital Projects & Infrastructure, 1-34.

Bikitsha, L., & Haupt, T. (2011). Impact of prefabrication on construction site health and safety: Perceptions of designers and contractors.

Bouleimen, K., & Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European journal of operational research, 149(2), 268-281.

Burkov, A. (2019). The hundred-page machine learning book (vol. 1): Andriy Burkov Canada.

CCQ. (2017). Statistiques Annuelles.

CEFRIO. (2011). Améliorer l'efficacité et la productivité du secteur de la construction grâce aux technologies de l'information. :

Cherkaoui, K. (2017). Planification tactique des grands projets d'ingénierie et de construction. (École Polytechnique de Montréal).

Choi, J. O., Chen, X. B., & Kim, T. W. (2019). Opportunities and challenges of modular methods in dense urban environment. International Journal of Construction Management. doi:10.1080/15623599.2017.1382093

Cottrell, W. D. (1999). Simplified program evaluation and review technique (PERT). Journal of construction Engineering and Management, 125(1), 16-22.

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. Communication présentée à International conference on computers and games (p. 72-83).

De Boer, R. (1998). Resource-constrained multi-project management. (PhD thesis, University of Twente, The Netherlands).

Elmaghraby, S. E. (1964). An algebra for the analysis of generalized activity networks. Management Science, 10(3), 494-514.

Gans, D., Corbusier, L., & Plattus, A. (2006). The Le Corbusier Guide : Princeton Architectural Press.

Gibb, A. G. (1999). Off-site fabrication: prefabrication, pre-assembly and modularisation : John Wiley & Sons.

Gibson Jr, G., Kaczmarowski, J., & Lore Jr, H. (1995). Preproject-planning process for capital facilities. Journal of construction engineering and management, 121(3), 312-318.

Guo, J.-X., Hu, C.-M., & Bao, R. (2019). Predicting the duration of a general contracting industrial project based on the residual modified model. KSCE Journal of Civil Engineering, 23(8), 3275-3284.

Hammad, A. W. A., Akbarnezhad, A., Rey, D., & Grzybowska, H. (2017). Decision support system for modular construction scheduling. Tiré de https://www.scopus.com/inward/record.uri?eid=2-s2.0-85032364560&partnerID=40&md5=5e997ed13876d9ec058ba982c444cde7

Hwang, B. G., Shan, M., & Looi, K. Y. (2018). Key constraints and mitigation strategies for prefabricated prefinished volumetric construction. Journal of Cleaner Production. doi:10.1016/j.jclepro.2018.02.136

Leon, G. P. d. (2011). Scheduling a Project at Different Levels.

Lepelley, D., Paul, M., & Smaoui, H. (2013). Introduction à la théorie des jeux (1): jeux non coopératifs. :

Li, L., Li, Z., Wu, G., & Li, X. (2018). Critical success factors for project planning and control in prefabrication housing production: A China study. Sustainability (Switzerland). doi:10.3390/su10030836

Lind, M. R., & Sulek, J. M. (2000). A methodology for forecasting knowledge work projects. Computers & Operations Research, 27(11-12), 1153-1169.

Lopez, D., & Froese, T. M. (2016). Analysis of costs and benefits of panelized and modular prefabricated homes. Procedia engineering, 145, 1291-1297.

Mikulakova, E., König, M., Tauscher, E., & Beucke, K. (2010). Knowledge-based schedule generation and evaluation. Advanced Engineering Informatics, 24, 389-403. doi:10.1016/j.aei.2010.06.010

Modular Building Institute. (2020). Prefabrication and modular construction 2020. :

Murtaza, M. B., Fisher, D. J., & Skibniewski, M. J. (1993). Knowledge-based approach to modular construction decision support. Journal of Construction Engineering and Management, 119(1), 115-130.

Pellerin, R., Perrier, N., & Berthaut, F. (2020). A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. European Journal of Operational Research, 280(2), 395-416.

Priddy, K. L., & Keller, P. E. (2005). Artificial neural networks: an introduction (vol. 68): SPIE press.

Relich, M., & Muszyński, W. (2014). The use of intelligent systems for planning and scheduling of product development projects. Procedia computer science, 35, 1586-1595.

Rosin, C. D. (2011). Multi-armed bandits with episode context. Annals of Mathematics and Artificial Intelligence, 61(3), 203-230.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., & Lanctot, M. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484-489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., & Bolton, A. (2017). Mastering the game of go without human knowledge. nature, 550(7676), 354-359.

Smith (réalisateur). (2018). Deep Learning for Complex Construction Sequencing Evann Smith (Bechtel Corporation). Tiré de https://vimeo.com/274802578

Song, J., Fagerlund, W. R., Haas, C. T., Tatum, C. B., & Vanegas, J. A. (2005). Considering prework on industrial projects. Journal of construction engineering and management, 131(6), 723-733.

Spangler, R. (2005). Front End Loading (FEL) and Process Engineering Workflow.

Speranza, M. G., & Vercellis, C. (1993). Hierarchical models for multi-project planning and scheduling. European Journal of Operational Research, 64(2), 312-325.

Sunke, N. (2008). Planning of construction projects: a managerial approach.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction : MIT press.

Turner, C. J., Oyekan, J., Stergioulas, L., & Griffin, D. (2021). Utilizing Industry 4.0 on the Construction Site: Challenges and Opportunities. IEEE Transactions on Industrial Informatics, 17(2), 746-756. doi:10.1109/TII.2020.3002197

Van Den Herik, H. J., Uiterwijk, J. W., & Van Rijswijck, J. (2002). Games solved: Now and in the future. Artificial Intelligence, 134(1-2), 277-311.

Van der Weijde, G. (2008). Front-End Loading in the Oil and Gas Industry.

Winge, S., Albrechtsen, E., & Mostue, B. A. (2019). Causal factors and connections in construction accidents. Safety science, 112, 130-141.

# ANNEXE A CODE DU MODÈLE

Cette annexe présente l'ensemble du code développé pour le modèle.

## MAIN.PY

```
import logging
import coloredlogs
from Coach import Coach
from Jeu import Jeu
from NNet import NNetWrapper as nn
import os
import sys
sys.setrecursionlimit(10**6)
from utils import *
import faulthandler


faulthandler.enable(file=sys.stdout,all_threads=False)
log = logging.getLogger(__name__)

coloredlogs.install(level='INFO')

args = dotdict({
    'numIters': 50,
    'numEps': 100,              # Number of complete self-play games to
simulate during a new iteration.
    'tempThreshold': 1,
    'updateThreshold': 0.51,     # During arena playoff, new neural net will
be accepted if threshold or more of games are won.
    'maxlenOfQueue': 200000,    # Number of game examples to train the neural
networks.
    'numMCTSSims': 20,          # Number of games moves for MCTS to simulate.
    'arenaCompare': 30,         # Number of games to play during arena play to
determine if new net will be accepted.
    'cpuct': 1,
    'checkpoint': './temp/',
    'load_model': False,
    'load_folder_file': ('/dev/models/8x100x50','best.pth.tar'),
    'numItersForTrainExamplesHistory': 20,

})


def main():
    log.info('Loading the Game')
    Type = ['Logement','Technique','Deplacement']
    Mode = ['Conventionnel','Panel','Modulaire']
    size = [2,3,3]

    g = Jeu(size,Type,Mode)

    log.info('Loading the Neural Network')
    nnet = nn(g)
```

```
    if args.load_model:
        log.info('Loading checkpoint "%s/%s"...', args.load_folder_file)
        nnet.load_checkpoint(args.load_folder_file[0],
args.load_folder_file[1])
    else:
        log.warning('Not loading a checkpoint!')


    log.info('Loading the Coach...')
    c = Coach(g, nnet, args)

    if args.load_model:
        log.info("Loading 'trainExamples' from file...")
        c.loadTrainExamples()

    log.info('Starting the learning process')
    c.learn()

if __name__ == "__main__":
    main()
```

## ARENA.PY

```
import logging
import copy
import numpy as np
from tqdm import tqdm
from DataGenerator import Data

from MCTS import MCTS

log = logging.getLogger(__name__)

class Arena():
    """
    An Arena class where any 2 agents can be pit against each other.
    """

    def __init__(self, player1, player2, game,i,args):
        """
        Input:
            game: Game object

        """
        self.player1 = player1
        self.player2 = player2
        if player3 != None:
            self.player3 = player3
        self.game = game
        self.display = display
        self.it = i
        self.scores = []
        self.args = args
```

```
    def playGame(self):
        """
        Executes one episode of a game.

        Returns:
            either
                winner: player who won the game (1 if player1, -1 if player2)
            or
                draw result returned from the game that is neither 1, -1, nor
0.
        """

        size = self.game.size
        data = Data(size[0],size[1],size[2])
        state = self.game.getInitGame(data)
        state2 = self.game.getInitGame(data)

        mcts1 = MCTS(self.game,self.player1,self.args)
        # Player 1
        while self.game.getGameEnded(state) == 0:

            pi1 = mcts1.getActionProb(state, temp=0)
            action = np.random.choice(len(pi1), p=pi1)
            state = self.game.getNextState(state, action)

        score1 = self.game.getGameEnded(state)
        self.scores.append(score1)


        mcts2 = MCTS(self.game,self.player2,self.args)
        # Player 2
        while self.game.getGameEnded(state2) == 0:

            pi2 = mcts2.getActionProb(state2, temp=0)
            action2 = np.random.choice(len(pi2), p=pi2)
            state2 = self.game.getNextState(state2, action2)

        score2 = self.game.getGameEnded(state2)

        if score1 < score2:
            return 1
        elif score1 == score2:
            return 0
        else:
            return -1

    def playGames(self, num):
        """
        Plays num games between player1 and player2.

        Returns:
            oneWon: games won by player1
            twoWon: games won by player2
            draws:  games won by nobody
        """

        oneWon = 0
```

```
        twoWon = 0
        draws = 0
        for _ in tqdm(range(num), desc="Arena.playGames"):
            gameResult = self.playGame(verbose=verbose)
            if gameResult == 1:
                oneWon += 1
            elif gameResult == -1:
                twoWon += 1
            else:
                draws += 1

        destFile = 'arena.txt'
        with open(destFile, 'a') as f:
            print(self.it,self.scores, file=f)
        self.it += 1


        return oneWon, twoWon, draws
```

## COACH.PY

```
import logging
import os
import sys
sys.setrecursionlimit(10000000)
from collections import deque
from pickle import Pickler, Unpickler
from random import shuffle
import numpy as np
from tqdm import tqdm
import keras
import copy
from Arena import Arena
from MCTS import MCTS
from DataGenerator import Data

log = logging.getLogger(__name__)

class Coach():
    """
    This class executes the self-play + learning. It uses the functions
defined
    in Game and NeuralNet. args are specified in main.py.
    """

    def __init__(self, game, nnet, args):
        self.game = game
        self.nnet = nnet
        self.pnet = self.nnet.__class__(self.game)  # the competitor network
        self.args = args
        self.mcts = MCTS(self.game, self.nnet, self.args)
        self.trainExamplesHistory = []  # history of examples from
args.numItersForTrainExamplesHistory latest iterations
```

```
        self.skipFirstSelfPlay = False  # can be overriden in
loadTrainExamples()

    def executeEpisode(self):
        """
        This function executes one episode of self-play.
        As the game is played, each turn is added as a training example to
        trainExamples. The game is played till the game ends. After the game
        ends, the outcome of the game is used to assign values to each example
        in trainExamples.

        Returns:
            trainExamples: a list of examples of the form (canonicalBoard,
pi,v)
                           pi is the MCTS informed policy vector, v is +1 if
                           the player eventually won the game, else -1.
        """
        trainExamples = []
        size = self.game.size
        data = Data(size[0],size[1],size[2])
        state = self.game.getInitGame(data)
        episodeStep = 0

        while True:
            episodeStep += 1
            temp = 1
            pi = self.mcts.getActionProb(state, temp=temp)

            trainExamples.append([state.lots, pi, None])

            action = np.random.choice(len(pi), p=pi)
            state = self.game.getNextState(state, action)

            r = self.game.getGameEnded(state)
            if r != 0:
                return [(x[0].tolist(), x[1], r) for x in trainExamples]

    def learn(self):
        """
        Performs numIters iterations with numEps episodes of self-play in each
        iteration. After every iteration, it retrains neural network with
        examples in trainExamples (which has a maximum length of
maxlenofQueue).
        It then pits the new neural network against the old one and accepts it
        only if it wins >= updateThreshold fraction of games.
        """

        for i in range(1, self.args.numIters + 1):
            log.info(f'Starting Iter #{i} ...')

            # examples of the iteration
            if not self.skipFirstSelfPlay or i > 1:
                iterationTrainExamples = deque([],
maxlen=self.args.maxlenOfQueue)
                for _ in tqdm(range(self.args.numEps), desc="Self Play"):
                    self.mcts = MCTS(self.game, self.nnet, self.args)  # reset
search tree
```

```
                    iterationTrainExamples += self.executeEpisode()
                # save the iteration examples to the history
                self.trainExamplesHistory.append(iterationTrainExamples)


            if len(self.trainExamplesHistory) >
self.args.numItersForTrainExamplesHistory:
                self.trainExamplesHistory.pop(0)

            # shuffle examples before training
            trainExamples = []
            for e in self.trainExamplesHistory:
                trainExamples.extend(e)
            shuffle(trainExamples)

            # training new network, keeping a copy of the old one
            self.nnet.save_checkpoint(folder=self.args.checkpoint,
filename='temp.pth.tar')
            self.pnet.nnet.model =
self.pnet.load_checkpoint(folder=self.args.checkpoint,
filename='temp.pth.tar')

            self.nnet.train(trainExamples)

            log.info('PITTING AGAINST PREVIOUS VERSION')
            arena = Arena(self.nnet,self.pnet, self.game,i,self.args)
            '''arena = Arena(lambda x: np.argmax(pmcts.getActionProb(x,
temp=0)),
                             lambda x: np.argmax(nmcts.getActionProb(x, temp=0)),
self.game,i)
            '''

            pwins, nwins, draws = arena.playGames(self.args.arenaCompare)

            log.info('NEW/PREV WINS : %d / %d ; DRAWS : %d' % (nwins, pwins,
draws))
            if pwins + nwins == 0 or float(nwins) / (pwins + nwins) <
self.args.updateThreshold:
                log.info('REJECTING NEW MODEL')
                self.nnet.load_checkpoint(folder=self.args.checkpoint,
filename='temp.pth.tar')
            else:
                log.info('ACCEPTING NEW MODEL')
                self.nnet.save_checkpoint(folder=self.args.checkpoint,
filename=self.getCheckpointFile(i))
                self.nnet.save_checkpoint(folder=self.args.checkpoint,
filename='best.pth.tar')

    def getCheckpointFile(self, iteration):
        return 'checkpoint_' + str(iteration) + '.pth.tar'

    def saveTrainExamples(self, iteration):
        folder = self.args.checkpoint
        if not os.path.exists(folder):
            os.makedirs(folder)
        filename = os.path.join(folder, self.getCheckpointFile(iteration) +
".examples")
```

```
        with open(filename, "wb+") as f:
            Pickler(f).dump(self.trainExamplesHistory)
        f.closed

    def loadTrainExamples(self):
        modelFile = os.path.join(self.args.load_folder_file[0],
self.args.load_folder_file[1])
        examplesFile = modelFile + ".examples"
        if not os.path.isfile(examplesFile):
            log.warning(f'File "{examplesFile}s" with trainExamples not
found!')
            r = input("Continue? [y|n]")
            if r != "y":
                sys.exit()
        else:
            log.info("File with trainExamples found. Loading it...")
            with open(examplesFile, "rb") as f:
                self.trainExamplesHistory = Unpickler(f).load()
            log.info('Loading done!')

            # examples based on the model were already collected (loaded)
            self.skipFirstSelfPlay = True
```

## JEU.PY

```
import numpy as np
import copy


class State():
    def __init__(self,data):
        '''
        Output = time, space, lots
        '''
        # Time
        self.time = 0

        # Size
        size = [0]*3 #[z,y,x]
        size[0] = data.z
        size[1] = data.y
        size[2] = data.x

        # Empty space
        self.space = np.zeros((size[0],size[1],size[2]))

        # List construction
        self.lots = np.zeros((data.nb_max_lots,11))

    def getInitState(self,data):

        nb_lot = len(data.liste)
```

```
        for i in range(nb_lot):
            self.lots[i] =
[data.liste[i,1],data.liste[i,2],data.liste[i,3],data.liste[i,4],data.liste[i,
5],data.liste[i,6],data.liste[i,7],data.liste[i,8],data.liste[i,9],0,-1]
            if data.liste[i,11]==[]:
                self.lots[i,9] = int(1) # To be planned
            else:
                self.lots[i,9] = int(0)


class Jeu():
    def __init__(self,size,Type,Mode):
        self.size = size
        self.Type = Type
        self.Mode = Mode


    def getInitGame(self,data):
        ''' Créer le State à partir des données data

        '''
        s = State(data)
        s.getInitState(data)
        self.data = data
        self.maxscorecout = 0
        self.maxscoreduree = 0
        for lot in s.lots:
            self.maxscorecout += lot[5]
            self.maxscoreduree += lot[6]
        return s

    def getNextState(self, state, choix):
        # Je suppose que le move est possible !!
        # Si un lot est choisi, on l'execute et on redonne le meme temps et la
mise a jour du space
        # Si aucun lot n'est choisi, on incrémente le temps
        ''' Retourne un objet State mis à jour suivant le choix du lot
effectué
        '''
        nb_lot = len(self.data.liste)
        n = self.getActionsize()

        if choix >= nb_lot:
            choix = n-1

        s = State(self.data)
        s.space = state.space.copy()
        s.lots = state.lots.copy()
        s.time = copy.copy(state.time)

        if choix == n-1:
            s.time += 1
            for k in range(s.space.shape[0]): #z
                for j in range(s.space.shape[1]): #y
                    for i in range(s.space.shape[2]): #x
                        if s.space[k][j][i] >= 1:
                            s.space[k][j][i] -= 1
```

```
        else:
            zonesup = self.getzonesup(choix,s)
            for zone in zonesup:
                s.space[zone[0]][zone[1]][zone[2]] = s.lots[choix][6]
            s.lots[choix,9] = -1
            s.lots[choix,10] = s.time
            if self.data.liste[choix,12] != []:
                for lot in self.data.liste[choix,12]:
                    s.lots[lot,9] = -1

        return self.majplanif(s)

    def majplanif(self,state):
        '''
        Met à jour la colonne 9 qui permet de savoir si le lot est planifiable
        '''
        s = State(self.data)
        s.lots = state.lots.copy()
        s.space = state.space.copy()

        n = len(self.data.liste)
        for idlot in range(n):
            if s.lots[idlot][9] != -1:
                prec = self.checkprecetage(idlot,s)
                if prec:
                    libre = self.checkspace(idlot,s)
                    if libre:
                        s.lots[idlot][9] = 1
                    else:
                        s.lots[idlot][9] = 0

        return s

    def checkprecetage(self,choix,state):
        ''' Retourne Vrai si les lots qui ont un lien de précédence avec choix
ont tous été planifiés
        Retourne Faux sinon
        '''

        listeprec = self.data.liste[choix,11]
        if listeprec == []:
            return True
        else:
            for idprec in listeprec:
                if (state.lots[idprec,9] == -1) and (state.lots[idprec,10] !=
-1):
                    return True
            return False

    def checkspace(self,idlot,state):
        ''' Retourne vrai si l'espace est libre pour le choix du lot
        '''
        s = State(self.data)
        s.lots = state.lots.copy()
        s.space = state.space.copy()
```

```
        zonesupl = self.getzonesup(idlot,state)
        for sup in zonesupl :
            etage = sup[0]
            while etage >= 0:
                if state.space[etage,sup[1],sup[2]] != 0 :
                    return False
                etage -= 1

        return True

    def getActionsize(self):
        # Retourne le nombre d'actions possibles pour ce projet :
x*y*z*NbMode+1(le +1 correspond à l'action de "passer")
        return int(self.size[0]*self.size[1]*self.size[2]*len(self.Mode)+1)

    def getzonesup(self,action,state):
        # Retourne une liste de coordonnées des emplacements de l'action ainsi
que la zone occupé en plus

        s = State(self.data)
        s.lots = state.lots.copy()
        s.space = state.space.copy()

        zoneplus = []
        z = int(s.lots[action,2])
        y = int(s.lots[action,3])
        x = int(s.lots[action,4])

        zoneplus.append([z,y,x])

        zonesup = self.data.liste[action,10]

        if zonesup == 1:
            # Bas
            if y+1 < self.size[1]:
                if s.space[z,y+1,x] == 0:
                    zoneplus.append([z,y+1,x])
                    return zoneplus

            # Haut
            if y-1 >= 0:
                if s.space[z,y-1,x] == 0:
                    zoneplus.append([z,y-1,x])
                    return zoneplus

            # Bas
            if x+1 < self.size[2]:
                if s.space[z,y,x+1] == 0:
                    zoneplus.append([z,y,x+1])
                    return zoneplus

            # Haut
            if x-1 >= 0:
                if s.space[z,y,x-1] == 0:
                    zoneplus.append([z,y,x-1])
                    return zoneplus
```

```
        return zoneplus

    def getGameEnded(self,state):
        n = len(self.data.liste)
        planif = 0
        cout = 0
        dureemax = -1
        for lot in state.lots:
            if lot[9] == -1:
                planif+=1
            if lot[10] != -1:
                if lot[10]>= dureemax:
                    dureemax = lot[10]
                cout += lot[5]

        if n == planif:
            score = -(cout/self.maxscorecout +
dureemax/self.maxscoreduree)/2.0
            return score
        else:
            return 0

    def getValidMoves(self,state):
        # return a fixed size binary vector
        valids = [0]*self.getActionsize()
        Lp = state.lots.copy()

        for i in range(len(Lp)):
            if Lp[i][9] == 1 :
                valids[i] = 1

        if sum(valids)==0:
            valids[-1] = 1
        return np.array(valids)

    def stringRepresentation(self, state):
        float_array = state.lots.astype(float)

        RETURN FLOAT_ARRAY.TOSTRING()
```

## MCTS.PY

```
import logging
import math
import numpy as np
EPS = 1e-8
log = logging.getLogger(__name__)

class MCTS():
    """
    This class handles the MCTS tree.
    """
```

```
    def __init__(self, game, nnet,args):
        self.game = game
        self.nnet = nnet
        self.args = args

        self.Qsa = {}  # stores Q values for s,a (as defined in the paper)
        self.Nsa = {}  # stores #times edge s,a was visited
        self.Ns = {}  # stores #times board s was visited
        self.Ps = {}  # stores initial policy (returned by neural net)

        self.Es = {}  # stores game.getGameEnded ended for board s
        self.Vs = {}  # stores game.getValidMoves for board s

        self.idstate = []


    def getActionProb2(self,state,temp):
        prob = [1.0/self.game.getActionsize()]*self.game.getActionsize()
        return prob


    def getActionProb(self, state, temp=1):
        """
        This function performs numMCTSSims simulations of MCTS starting from
        state.

        Returns:
            probs: a policy vector where the probability of the ith action is
                   proportional to Nsa[(s,a)]**(1./temp)
        """

        for i in range(self.args.numMCTSSims):
            self.search(state)


        s = self.getIdState(state)
        counts = [self.Nsa[(s, a)] if (s, a) in self.Nsa else 0 for a in
range(self.game.getActionsize())]

        if temp == 0:
            bestAs = np.array(np.argwhere(counts == np.max(counts))).flatten()
            bestA = np.random.choice(bestAs)
            probs = [0] * len(counts)
            probs[bestA] = 1.0
            return probs

        counts = [x ** (1. / temp) for x in counts]
        counts_sum = float(sum(counts))
        probs = [x / counts_sum for x in counts]
        return probs

    def search(self, state):
        """
        This function performs one iteration of MCTS. It is recursively called
        till a leaf node is found. The action chosen at each node is one that
        has the maximum upper confidence bound as in the paper.
```

```
        Once a leaf node is found, the neural network is called to return an
        initial policy P and a value v for the state. This value is propagated
        up the search path. In case the leaf node i s a terminal state, the
        outcome is propagated up the search path. The values of Ns, Nsa, Qsa
are
        updated.

        NOTE: the return values are the value of the current
        state.

        Returns:
            v: the value of the current state
        """
        s = self.getIdState(state)

        if s not in self.Es:
            self.Es[s] = self.game.getGameEnded(state)
        if self.Es[s] != 0:
            # terminal node
            return self.Es[s]

        if s not in self.Ps:
            # leaf node
            self.Ps[s], v = self.nnet.predict(state)
            valids = self.game.getValidMoves(state)
            self.Ps[s] = self.Ps[s] * valids  # masking invalid moves
            sum_Ps_s = np.sum(self.Ps[s])
            if sum_Ps_s > 0:
                self.Ps[s] /= sum_Ps_s  # renormalize
            else:
                self.Ps[s] = self.Ps[s] + valids

                self.Ps[s] /= np.sum(self.Ps[s])

            self.Vs[s] = valids
            self.Ns[s] = 0
            return v

        valids = self.Vs[s]
        cur_best = -float('inf')
        best_act = -1

        # pick the action with the highest upper confidence bound

        for a in range(self.game.getActionsize()):
            if valids[a]:
                if (s, a) in self.Qsa:
                    u = self.Qsa[(s, a)] + self.args.cpuct * self.Ps[s][a] *
math.sqrt(self.Ns[s]) / (1 + self.Nsa[(s, a)])
                else:
                    u = self.args.cpuct * self.Ps[s][a] * math.sqrt(self.Ns[s]
+ EPS)

                if u > cur_best:
                    cur_best = u
                    best_act = a
```

```
        a = best_act
        next_s = self.game.getNextState(state, a)

        v = self.search(next_s)

        if (s, a) in self.Qsa:
            self.Qsa[(s, a)] = (self.Nsa[(s, a)] * self.Qsa[(s, a)] + v) /
(self.Nsa[(s, a)] + 1)
            self.Nsa[(s, a)] += 1

        else:
            self.Qsa[(s, a)] = v
            self.Nsa[(s, a)] = 1

        self.Ns[s] += 1

        return v
```

## MODELE.PY

```
import numpy as np


class Random():
    def __init__(self, game):
        self.game = game

    def getActionProb(self, state,temp=0):
        valids = self.game.getValidMoves(state)
        somme = sum(valids)
        if somme == 0:
            probs = [0]*self.game.getActionsize()
            probs[-1] = 1
            return probs
        else:
            probs = [x/somme for x in valids]
        return probs


class UniMode():
    def __init__(self, game):
        self.game = game


    def getActionProb(self, state,temp=0):
        mini = np.Inf
        for lot in state.lots:
            if lot[1] != 0:
                lot[9] = -1

        valids = self.game.getValidMoves(state)
        somme = sum(valids)

        probs = [0]*self.game.getActionsize()
```

```
        imini = np.Inf
        for i in range(len(state.lots):
            if state.lots[i][6] < mini :
                mini = state.lots[i][6]
                imini = i

        if imini < np.Inf:
            probs[imini] = 1.0

        return probs
```

## PIT.PY

```
import Arena
from MCTS import MCTS
from Jeu import Jeu
from NNet import NNetWrapper as NNet
from Players import *
import os
import numpy as np
from utils import *
import sys
sys.setrecursionlimit(100000)

args = dotdict({
    'numIters': 50,
    'numEps': 50,              # Number of complete self-play games to
simulate during a new iteration.
    'tempThreshold': 15,        #
    'updateThreshold': 0.6,     # During arena playoff, new neural net will be
accepted if threshold or more of games are won.
    'maxlenOfQueue': 200000,    # Number of game examples to train the neural
networks.
    'numMCTSSims': 25,          # Number of games moves for MCTS to simulate.
    'arenaCompare': 10,         # Number of games to play during arena play to
determine if new net will be accepted.
    'cpuct': 1,

    'checkpoint': './temp/',
    'load_model': False,
    'load_folder_file': ('/dev/models/8x100x50','best.pth.tar'),
    'numItersForTrainExamplesHistory': 20,

})

Type = ['Logement','Technique','Deplacement']
Mode = ['Conventionnel','Panel','Modulaire']
size = [2,3,3]

g = Jeu(size,Type,Mode)

# all players
rp = Random(g)
up = UniMode(g)
```

```
i = 0
player2 = rp #random model
player3 = up #unimode model

num = 3

resultat = []

for model in os.listdir('./temp/'):
    n1 = NNet(g)
    n1.load_checkpoint(folder='./temp/', filename=model)
    player1 = n1

    arena = Arena.Arena(player1, player2, g,i,args,player3=player3)
    result = arena.playGamesTest(num)

    print(result)

    p1 = 100*result[0]/num
    p2 = 100*result[3]/num

    resultat.append((p1,p2))
    print('Le NN gagne à {} % contre le Random'.format(p1))
    print('Le NN gagne à {} % contre le Unimode'.format(p2))

print(resultat)
```

## DATAGENERATOR.PY

```
import numpy as np
import csv
import pandas as pd

class Data():

    def __init__(self,zmax,ymax,xmax):
        self.x = xmax
        self.y = ymax
        self.z = zmax
        self.liste = []


        header = ['ID','Type','Mode','Z','Y','X','Cout','Duree','Zone
sup','Prec','XOR']
        Type = ['Logement','Technique','Deplacement']
        Mode = ['Conventionnel','Panel','Modulaire']
        nb_type = len(Type)

        self.nb_max_lots = int(self.x*self.y*self.z*len(Mode)+1)

        possible = self.getTableauRef('mode_possible.csv')
        zonesup = self.getTableauRef('zonesup.csv')
        duree = self.getTableauRef('duree_type_mode.csv')
        cout = self.getTableauRef('cout_type_mode.csv')
```

```
        nid = 0
        for i in range(self.z): # z
            for j in range(self.y):# y
                for k in range(self.x):# x

                    # Choix type aléatoire
                    typetemp = np.random.randint(nb_type)

                    for m in range(len(possible[typetemp])):
                        if possible[typetemp][m] == 1: # Si c'est un mode
possible pour ce type
                            newrow = [0] * len(header)
                            # ID
                            newrow[0] = nid

                            # Type
                            newrow[1] = typetemp

                            # Mode
                            newrow[2] = m

                            # Zone
                            newrow[3] = i #z
                            newrow[4] = j #y
                            newrow[5] = k #x

                            # Zone sup
                            newrow[10] = int(zonesup[typetemp][m])

                            # Durée
                            newrow[7] = duree[typetemp][m]

                            # Cout
                            newrow[6] = cout[typetemp][m]

                            self.liste.append(newrow)
                            nid += 1


        # Precedence

        for i in range(len(self.liste)) :
            self.liste[i][11] = self.getIDdirection(i,-1,0,0)

        # XOR
        for i in range(len(self.liste)) :
            self.liste[i][12] = self.getIDxor(i)

        self.liste = np.array(self.liste)

    def getIDdirection(self,IDbase,z,y,x):
        """ Retourne une liste avec l'id du lot correspondant à IDbase + le
déplacement suivant x,y,z
        """
        liste = []
```

```
        zbase = self.liste[IDbase][3]
        ybase = self.liste[IDbase][4]
        xbase = self.liste[IDbase][5]

        if zbase >= 1:
            for i in range(len(self.liste)) :
                ztemp = self.liste[i][3]
                ytemp = self.liste[i][4]
                xtemp = self.liste[i][5]
                if (ztemp,ytemp,xtemp) == (zbase+z,ybase+y,xbase+x):
                    liste.append(i)

        return liste

    def getIDxor(self,IDbase):
        """ Permet de retourner les ID des lots non compatibles car c'est le
meme type mais des modes différents
        """
        liste = []
        typebase = self.liste[IDbase][1]
        modebase = self.liste[IDbase][2]
        zbase = self.liste[IDbase][3]
        ybase = self.liste[IDbase][4]
        xbase = self.liste[IDbase][5]

        for i in range(len(self.liste)):
            typetemp = self.liste[i][1]
            modetemp = self.liste[i][2]
            ztemp = self.liste[i][3]
            ytemp = self.liste[i][4]
            xtemp = self.liste[i][5]
            if (typebase,zbase,ybase,xbase) == (typetemp,ztemp,ytemp,xtemp)
and modebase != modetemp :
                liste.append(i)

        return liste

    def getTableauRef(self,file):
        e = np.genfromtxt(file, delimiter=",",skip_header=1)
        mask = np.ones(len(e[0]), dtype=bool)
        mask[0] = False
        result = []
        for i in range(len(e)):
            result.append(e[i][mask,...])
        result = np.array(result)

        return result
```

## NN_FUNCTIONS

```
import argparse
import os
import shutil
import time
import random
import numpy as np
import math
import sys
sys.path.append('../..')
from utils import *
import keras
import copy
import argparse
from NeuralNet import NeuralNet
from ConstructionNNet import ConstructionNNet as onnet
from matplotlib import pyplot as plt


args = dotdict({
    'lr': 0.001,
    'dropout': 0.3,
    'epochs': 10,
    'batch_size': 64,
    'cuda': False,
    'num_channels': 512,
})

class NNetWrapper(NeuralNet):
    def __init__(self, game):
        self.nnet = onnet(game, args)
        self.action_size = game.getActionsize()
        self.i = 1

    def preparation(self,examples):

        n = len(examples)

        input_boards, target_pis, target_vs = list(zip(*examples))
        input_boards = np.array(input_boards)
        input_boards = input_boards.astype('float64')

        target_pis = np.asarray(target_pis)
        target_vs = np.asarray(target_vs)
        target_vs = np.reshape(target_vs,(n,1))

        return input_boards,target_pis,target_vs


    def train(self, examples):
        """
        examples: list of examples, each example is of form (state, pi, v)
        """
        input_boards,target_pis,target_vs = self.preparation(examples)

        history = self.nnet.model.fit(x = input_boards, y = [target_pis,
target_vs], batch_size = args.batch_size, epochs = args.epochs)
```

```
        destFile = 'Loss_nn.txt'
        with open(destFile, 'a') as f:
            print(self.i,history.history['loss'], file=f)

        self.i += 1



    def predict(self, state):

        # preparing input
        liste = np.copy(state.lots)
        time = copy.copy(state.time)
        space = np.copy(state.space)

        trainstate = liste.astype(float)
        trainstate = trainstate[np.newaxis, :, :]

        trainX = np.array(trainstate)
        # run
        pi, v = self.nnet.model.predict(trainX)

        espace_vide = np.zeros_like(space)

        if time == 0 and np.array_equal(space,espace_vide):
            n = len(liste)
            val = 1./n

            mask1 = np.array([1]*n)
            mask2 = np.array([0]*(self.action_size-n))
            mask = np.hstack([mask1,mask2])
            pi[0] = np.array(val*mask)

        return pi[0], v[0]


    def save_checkpoint(self, folder='checkpoint',
filename='checkpoint.pth.tar'):
        filepath = os.path.join(folder, filename)
        if not os.path.exists(folder):
            print("Checkpoint Directory does not exist! Making directory
{}".format(folder))
            os.mkdir(folder)
        else:
            print("Checkpoint Directory exists! ")
        keras.models.save_model(
            self.nnet.model,
            filepath,
            overwrite=True,
            include_optimizer=True,
            save_format=None,
            signatures=None,
            options=None
        )

    def load_checkpoint(self, folder='checkpoint',
filename='checkpoint.pth.tar'):
```

```
        filepath = os.path.join(folder, filename)
        if not os.path.exists(filepath):
            raise("No model in path {}".format(filepath))

        return keras.models.load_model(
            filepath,
            custom_objects=None,
            compile=True

        )
```

## NEURALNET.PY

```
import sys
sys.path.append('..')
import argparse
from keras.models import *
from keras.layers import *
from keras.optimizers import *
from keras import initializers
import tensorflow as tf
import numpy as np

class ConstructionNNet():
    def __init__(self, game,args):
        # game params
        self.action_size = game.getActionsize()
        self.size = game.size
        lr = 0.001
        self.dropout = 0.3
        self.epochs = 10
        self.batch_size = 64
        self.cuda = False
        self.num_channels = 512

        # Neural Net

        self.input = Input(shape=(self.action_size,11))

        x = Dense(64, activation="relu")(self.input)
        x = Flatten()(x)

        initializer_pi = initializers.Ones()
        self.pi = Dense(self.action_size, activation='softmax', name='pi',
kernel_initializer=initializer_pi)(x)   # batch_size x self.action_size


        self.v = Dense(1, activation='relu', name='v')(x)    # batch_size x 1

        self.model = Model(inputs=self.input, outputs=[self.pi, self.v])


self.model.compile(loss=['categorical_crossentropy','mean_squared_error'],
optimizer=Adam(lr))
```