Institut National des Sciences Appliquées et des Technologies

UNIVERSITE DE CARTHAGE

STAGE INGÉNIEUR

Génie Logiciel

RN quantifié pour un contexte deep learning embarqué

Auteur:

Rami ZOUARI

2021/2022

Résumé

Les réseaux de neurones sont devenus très utilisés dans l'intelligence artificielle grâce à leurs performances de prédictions.
Ces réseaux de neurones sont parfois très complexes, et il n'est pas pratique de les utiliser dans des systèmes à ressources limités.

Pour cela, dans ce rapport nous allons introduire les réseaux de neurones binaires (BNNs), les formaliser, étudier quelques modèles binarisés et dériver leurs formules, implémenter les résultats trouvés dans une bibliothèque qu'on va nommer **binaryflow**.
Et finalement, nous allons implémenter et entraîner des modèles binarisés pour la classification et régression, et nous allons comparer les performances de prédiction, ainsi que les performances temps et mémoire de ces BNNs et leurs contrepart classique.

Remerciement

Je veux remercier mes encadreurs:

- Mme. Meriem JAÏDANE
- Mme. Yousra BEN JEMÂA
- Mme. Rim AMARA BOUJEMAA
- Mr. Nader MECHERGUI

ainsi que toutes l'équipe de **dB Sense**, qui m'ont donné la chance de travailler sur ce projet intéressant, et leur expertise a été extrêmement précieuse dans la formulation des questions de recherche et de la méthodologie
Vos commentaires perspicaces m'ont poussés à affiner ma réflexion et ont fait passer le travail au niveau supérieur.
J'ai eu la chance de quitter ma zone de confort en travaillant sur ce sujet original, et je m'en suis très reconnaissants.

Contenu

Rami ZOUARI
INSAT, Tunisia

Rami ZOUARI
INSAT, Tunisia

Liste des Figures

Liste des Tableaux

List of Algorithms

Introduction

Avec l'explosion de l'intelligence artificielle, et surtout les modèles d'apprentissage profonds, la complexité des modèles a subi une croissance considérable, qui les rend inexploitable dans les systèmes à complexité limitée.

Dans ce rapport, nous allons étudier la quantification des paramètres et des entrées des couches du réseau de neurones sur un seul bit.

Ce rapport va détailler notre approche de la formalisation des BNNs, de l'analyse et généralisation des approches existantes, vers l'implémentation d'une bibliothèque unifiant les BNNs, et son utilisation. Il est composé de 6 chapitres.

Dans le premier chapitre, nous allons présenter la societé **dB Sense** et sa méthodologie.

Dans le deuxième chapitre, nous allons donner une petite histoire de l'apprentissage profond, et puis poser le problème de la grande complexitée de ces modèle, en posant la binarisation comme une solution.

Dans le troisième chapitre, nous allons formaliser notre approche, en définissant les BNNs. Après nous allons poser quelques problèmes dans l'entraînement. Après nous allons proposer les optimisations temps et mémoire qu'on peut exploiter avec les BNNs.
Finalement, nous allons proposer l'algorithme d'entraînement et d'interférence des BNNs.

Dans le quatrième chapitre, nous allons étudier et analyser quelques BNNs répandus dans la littérature, en conformant avec notre définition proposée.

Dans le cinquième chapitre, nous allons implémenter la bibliothèque **binaryflow**, en justifiant les paradigmes utilisés.

Dans le sixième chapitre, nous allons utiliser les différents BNNs étudiés sur 3 jeux de données.
Pour chacun de ces modèles nous allons analyser son performance de prédiction, sa taille mémoire au déploiement, et une estimation sur la complexité de son interférence en calculant le nombres d'instruction équivalents.

Chapter 1

Cadre du Stage

Introduction

Chapter 2

Analysis & Implementation

2.1 Introduction

2.2 Formalisation

To define a Mean Payoff Game, we will start by formalising a weighted di-graph[1].

2.2.1 Di-Graph

A Weighted Di-Graph $\mathcal{G}$ is a tuple $(\mathcal{V}, \mathcal{E}, \mathcal{W})$ where:

- $\mathcal{V}$ is the set of vertices.
- $\mathcal{E} \subseteq V \times V$ is the set of edges.
- $\mathcal{W} : \mathcal{E} \to \mathbb{G}$ is the weight function, assigning a weight for every edge, with $\mathbb{G}$ some ordered abelian group[2].

2.2.2 Mean Payoff Game

Formally, a **Mean Payoff Graph** is a tuple $(\mathcal{V}, \mathcal{E}, \mathcal{W}, \mathcal{P}, s, p)$ where:

- $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ is a di-graph.
- $s \in \mathcal{V}$ denotes the starting position.
- $\mathcal{P} = \{\text{Max}, \text{Min}\}$ is the set of players.
- $p \in \mathcal{P}$ the starting player

A **Mean Payoff Game** is a perfect information, zero-sum, turn based game played indefinitively on a Mean Payoff Graph as follow:

- The game starts at $u_0 = s$, with player $p_0 = p$ starting.
- For each $n \in \mathbb{N}$, Player p_n will choose a vertex $u_{n+1} \in \text{Adj}\, u_n$, with a payoff $w_n = W(u_n, u_{n+1})$
- The winner of the game will be determined by the Mean Payoff. There are different winning conditions.

Name	Max winning criteria	Min winning criteria	Draw criteria
C_1	$\liminf_{n\in\mathbb{N}*} \frac{1}{n}\sum_{k=0}^{n-1} w_k >\geqslant 0$	$\liminf_{n\in\mathbb{N}*} \frac{1}{n}\sum_{k=0}^{n-1} w_k < 0$	
C_2	$\liminf_{n\in\mathbb{N}*} \frac{1}{n}\sum_{k=0}^{n-1} w_k > 0$	$\liminf_{n\in\mathbb{N}*} \frac{1}{n}\sum_{k=0}^{n-1} w_k < 0$	$\liminf_{n\in\mathbb{N}*} \frac{1}{n}\sum_{k=0}^{n-1} w_k = 0$
C_3	$\liminf_{n\in\mathbb{N}*} \frac{1}{n}\sum_{k=0}^{n-1} w_k > 0$	$\limsup_{n\in\mathbb{N}*} \frac{1}{n}\sum_{k=0}^{n-1} w_k < 0$	$\begin{cases} \liminf_{n\in\mathbb{N}*} \frac{1}{n}\sum_{k=0}^{n-1} w_k \leqslant 0 \\ \limsup_{n\in\mathbb{N}*} \frac{1}{n}\sum_{k=0}^{n-1} w_k \geqslant 0 \end{cases}$

Table 2.1: Winning conditions for Mean Payoff Games

Here, table 2.1 gives the different winning criteria that we have considered:

- C_1 was used in [3] to calculate the optimal strategy for player Max
- C_2 is the dual of C_1.
- C_3 is symmetric[3], and will be used for our machine learning model.

2.2.3 Well Foundness

It is not very clear from the definition that the game is well founded.
In fact, there are choices for which the mean payoff does not converge. That is the sequence $\left(\frac{1}{n}\sum_{k=0}^{n-1} w_k\right)_{n\in\mathbb{N}*}$ does not converge.
One such example is the sequence defined by:

$$w_n = (-1)^{\lfloor \log_2(n+1) \rfloor}$$

[1]Directed Graph
[2]This definition is too general. We will only consider $\mathbb{G} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$. Also, $\mathbb{G}$ itself should be clear from the context.
[3]It does not give an advantage towards any player.

Rami ZOUARI
INSAT, Tunisia

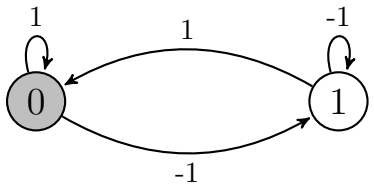
For that sequence, the $(2^r - 1)$-step mean payoff is equal to:

$$\begin{aligned}\sum_{k=0}^{2^r-2} w_k &= \sum_{k=1}^{2^r-1} (-1)^{\lfloor \log_2(k) \rfloor} = \sum_{i=0}^{r-1} \sum_{j=2^i}^{2^{i+1}-1} (-1)^{\lfloor \log_2(j) \rfloor} \\ &= \sum_{i=0}^{r-1} \sum_{j=2^i}^{2^{i+1}-1} (-1)^i = \sum_{i=0}^{r-1} (2^{i+1} - 2^i)(-1^i) \\ &= \sum_{i=0}^{r-1} (-2)^i = \frac{1-(-2)^r}{3} \\ \implies \frac{1}{2^r-1} \sum_{k=0}^{2^r-2} w_k &= \frac{1}{3} \cdot \frac{1-(-2)^r}{2^r-1} = \frac{1}{3} \cdot \frac{2^{-r} - (-1)^r}{1-2^{-r}}\end{aligned}$$

That sequence has two accumulation points $\pm\frac{1}{3}$, and thus, it does not converge.

On the other hand, the introduction of the supremum and infimum operators will solve the convergence problem, as the resulting sequences will become monotone.

An example of an execution that gives a rise to such payoffs is the following Meab Payoff Game instance[4]:



(a) Representation of the Mean Payoff Game

Pair of strategies defined as:

$$\begin{aligned}\Phi : & V^+ \times P \to V \\ & (s_0 \dots s_r, p) \to B(r) \bmod 2\end{aligned}$$

With $B(r)$ the position of the left-most bit in the binary representation of r

(b) Definition of both strategies

Figure 2.1: An example of an execution with non-convergent Mean Payoffs

2.2.4 Properties

A Mean Payoff Game has many properties that are interesting from a game theory perspective.

Two Player

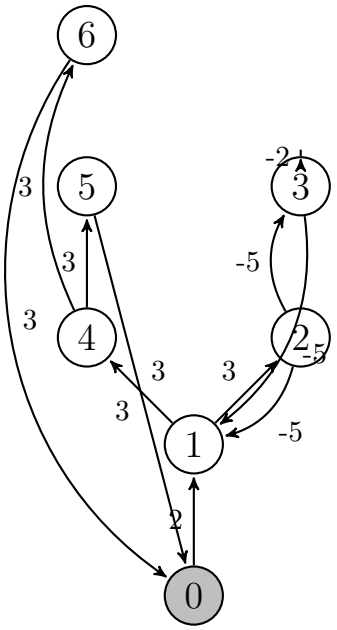
The game is a two player game.

Turn Based

2.2.5 Symmetries

Mean Payoff Games exhibits many natural symmetries.

[4]Note that the proposed pair of strategies is odd in the sense that it appears that both players cooperated on the construction of non-convergent mean payoffs instead of trying ot win the game.



Duality

The main symmetry is the duality between Max and Min .
For this we will define the dual $\bar{G}$ of a mean payoff game $G = (V, E, WP, s, p)$ as the following:

$$\bar{G} = (V, E, -W, s, \bar{p})$$

This duality is important due to the following theorem.

Theorem 1 *For every mean payoff game G, the objective of player* Max *is equivalent to the objective of player* Min *in* $\bar{G}$.

With that, there is not any major difference between Max and Min from a theoretical point of view. In fact, without any loss of generality, we can assume that Max is the starting player. And this is what we will do by default in this report.

2.2.6 Strategy

Deterministic Strategies

Let p be a player.
A (deterministic) strategy is a function $\Pi^p : V^+ \to V$ such that:

$$\forall v_0 \dots v_r \in V^+, \quad \Pi_p(v_0 \dots v_r) \in \mathrm{Adj}\, v$$

If the strategy does only depend on the current vertex, we say it is a memoryless (deterministic) strategy. $\Pi : V \to V$
In this report, we will use the term positional strategies as an alias for memoryless deterministic strategies, which is conforming to the established litterature of mean payoff games.
Positional strategies are crucial for our analysis as a result of the following theorem.

Theorem 2 *For all Mean Payoff Games, each player has an optimal positional strategy.*

Probabilistic Strategies

A probabilistic strategy is a random process that assigns for each sequence of vertices $v \in \mathcal{V}$ a probability distribution over $\operatorname{Adj} v$. This constitutes the most general strategy of a player:

$$\forall v_0 \dots v_r \in V^+, \quad \Pi_p(v_0 \dots v_r) \in \mathscr{D}(\operatorname{Adj} v)$$

Considered Strategies

Strategies that depends in complete past histories are in general intractable. For Mean Payoff Game, it is proven that the optimal strategy is a **deterministic** and **memoryless**.
For that we will only consider **memoryless** strategies. And for the scope of this report:

- A deterministic strategy should refer to memoryless deterministic strategy.
- A probabilistic strategy should refer to memoryless probabilistic strategy.
- A strategy should refer to memoryless deterministic strategy.

We will still consider (memoryless) probabilistic strategies as they reside in a smooth space, and thus they can be used for machine learning purposes.

Deterministic Optimal Strategy

There are three kinds of optimality:

Weak Optimality : In the deterministic case, a strategy Φ of player $p \in P$ is weakly optimal if one of the following is true:

- For each strategy Φ^p of player p, player $\bar{p}$ can win the game by finding a countering strategy $\Phi^{\bar{p}}$.
- Player p will not lose the game no matter his opponent's strategy

Strong Optimality : In the deterministic case, a strategy Φ of player $p \in P$ is strongly optimal if one of the following is true:

- For each strategy Φ^p of player p, player $\bar{p}$ can win or tie the game by finding a countering strategy $\Phi^{\bar{p}}$.
- Player p will win the game no matter his opponent's strategy

Payoff Optimality : In the deterministic case, a strategy Φ of player $p \in P$ is payoff optimal if independently of $\bar{p}$'s strategy it:

- Maximises the Mean Payoff if $p = \text{Min}$
- Minimises the Mean Payoff otherwise

Now we have the following hiearchy considering the set of optimal strategies:

$\forall$Mean Payoff Game $G, \forall p \in P, \quad \texttt{PayoffOptimal}(G, p) \subseteq \texttt{StrongOptimal}(G, p) \subseteq \texttt{WeakOptimal}(G, p)$

Rami ZOUARI
INSAT, Tunisia

2.2.7 Mean Payoff

We have used the word "Mean Payoff" extensively, and they are the central entity in mean payoff games[5] but we still did not define it.
We had to delay the definition as it requires the knowledge of the mechanics of the game, and how strategies work. This section will define and formalize the mean payoff, and highlights its relevance.

Mean Payoff

For a mean payoff game G, with a deterministic pair of strategies $(\Phi^{\text{Max}}, \Phi^{\text{Min}})$, we will define two terms $v^+(G, \Phi^{\text{Max}}, \Phi^{\text{Min}})$ and $v^-(G, \Phi^{\text{Max}}, \Phi^{\text{Min}})$ as follow[6]:

$$v^+(G, \Phi^{\text{Max}}, \Phi^{\text{Min}}) = \limsup_{n \in \mathbb{N}*} \frac{1}{n} \sum_{k=0}^{n-1} w_k$$

$$v^-(G, \Phi^{\text{Max}}, \Phi^{\text{Min}}) = \liminf_{n \in \mathbb{N}*} \frac{1}{n} \sum_{k=0}^{n-1} w_k$$

Where $(w_n)_{n \in \mathbb{N}}$ is the sequence of payoffs generated by the instance.
These two terms were used in table 2.1 when discussing the winning conditions, and we will call them respectively the supremum mean payoff, and the infimum mean payoff.

Theorem 3 *For every mean payoff game G, and every pair of strategies $(\Phi^{\text{Max}}, \Phi^{\text{Min}})$, both the supremum mean payoff $v^+(G, \Phi^{\text{Max}}, \Phi^{\text{Min}})$ and the infimum mean payoff $v^-(G, \Phi^{\text{Max}}, \Phi^{\text{Min}})$ are guaranteed to exist, and:*

$$v^-(G, \Phi^{\text{Max}}, \Phi^{\text{Min}}) \leqslant v^+(G, \Phi^{\text{Max}}, \Phi^{\text{Min}}) \tag{2.1}$$

If both terms are equal in equation (2.1), we say that the game instance has a mean payoff $v(G, \Phi^{\text{Max}}, \Phi^{\text{Min}}) = v^-(G, \Phi^{\text{Max}}, \Phi^{\text{Min}}) = v^+(G, \Phi^{\text{Max}}, \Phi^{\text{Min}})$.

Convergence Issues

This mean payoff itself is the heart of mean payoff games. A major problem lies in the fact that the existence of a mean payoff is not guaranteed, and an example for such case is found in the figure 2.1.
Now, fortunately, this is not an issue, as we are interested in good strategies. Theorem 2 proven in [11] states that for each player, the set of optimal strategies contains a positional one. This is a very important result as we can limit the domain of our optimization problem[7] to the more tractable positional[8] and behavioural[9] strategies.
We start by tackling the existence of the mean payoff for positional strategies in the the following theorem:

Theorem 4 *For every mean payoff G, and a pair of positional strategies $(\Phi^{\text{Max}}, \Phi^{\text{Min}})$, the mean payoff $v(G, \Phi^{\text{Max}}, \Phi^{\text{Min}})$ exists.*

[5] This explains the name "mean payoff game".
[6] For probabilistic strategies, both terms are random variables, so we will be interested in their expected values.
[7] The problem of finding optimal strategies
[8] memoryless and deterministic
[9] memoryless and probabilistic

We will give a constructive proof of theorem 4 in section 2.3. We will also provide a linear algorithm for its calculation.
Furthermore, in our machine learning model, we will approach the problem using behavioural strategies. Also luckily, its existence is guaranteed by the following theorem:

Theorem 5 *For every mean payoff G, and a pair of behavioural strategies $(\Pi^{\text{Max}}, \Pi^{\text{Min}})$, the expected mean payoff $\mathbb{E}[v(G, \Pi^{\text{Max}}, \Pi^{\text{Min}})]$ exists.*

Unlike theorem 4 theorem 5 was very challenging to prove, and we were not able to find a direct proof in the literature. We had some probabilistic arguments that affirmed the result for almost all mean payoff games. This alone was enough for us to use it as a metric for probabilistic strategies. Eventually, we were able to give a formal proof, which is detailed in the appendix C.

Game Value

By combining both theorems 2 and 4, there is a mean payoff $v(G)$ that both players are guaranteed to achieve if they play optimally:

$$\begin{aligned} &\exists \Phi^{\text{Max}}, \forall \Phi^{\text{Min}}, \quad v^-(G, \Phi^{\text{Max}}, \Phi^{\text{Min}}) \geqslant v(G) \\ &\exists \Phi^{\text{Min}}, \forall \Phi^{\text{Max}}, \quad v^+(G, \Phi^{\text{Max}}, \Phi^{\text{Min}}) \leqslant v(G) \end{aligned}$$

Such mean payoff is called the value of the game. This value determines the winner of the game assuming both players play optimally.

2.3 Evaluating Strategies

Suppose we have a pair of potentially probabilitic strategies $(\Phi^{\text{Max}}, \Phi^{\text{Min}})$. The problem is to evaluate the winner without doing an infinite simulation of the game.

2.3.1 Monte-Carlo Simulations

This is the most intuitive evaluation method

2.3.2 Positional Strategies

If both strategies are deterministic and memoryless. Then the generated sequence of vertices $(s_n)_{n\in\mathbb{N}}$ will be completely determined by the recurrence relation:

$$s_n = \begin{cases} s & \text{if } n = 0 \\ \Phi^{\text{Max}}(s_{n-1}) & \text{if } n \text{ is odd} \\ \Phi^{\text{Min}}(s_{n-1}) & \text{otherwise} \end{cases}$$

This can be represented in the compact form:

$$\forall n \in \mathbb{N}^*, \quad (s_n, p_n) = (\Phi^{p_{n-1}}(s_{n-1}), \bar{p}_{n-1}) = F(s_{n-1}, p_{n-1})$$

Since $\mathcal{V} \times \mathcal{P}$ is a finite set and F is a function, such sequence will be eventually periodic, that is:

$$\exists N \in \mathbb{N}, \exists T \in \mathbb{N}^* / \quad \forall n \in \mathbb{N}_{\geqslant N}, \quad (s_n, p_n) = (s_{n+T}, p_{n+T})$$

Rami ZOUARI
INSAT, Tunisia

We can calculate its eventual period using the turtle hare algorithm.

Now, the mean payoff will be equal to the mean of weights that appears on the cycle. This can be proven as follow.:

$$\begin{aligned}
S_{aT+b+N} &= \sum_{k=0}^{aT+b+N-1} w_k \\
&= \sum_{k=0}^{N-1} w_k + \sum_{k=0}^{aT+b-1} w_{k+N} \\
&= \sum_{k=0}^{N-1} w_k + a\sum_{r=0}^{T-1} w_{r+N} + \sum_{r=0}^{b-1} w_{r+N} \\
\implies \left| S_{n+N} - \lfloor \frac{n}{T} \rfloor \sum_{r=0}^{T} w_{k+N} \right| &\leqslant (N+T-1) \max_{(u,v)\mathcal{E}} |\mathcal{W}(u,v)| \\
&\leqslant (N+T-1)\|\mathcal{W}\|_\infty \\
\implies \left| \frac{1}{n+N} S_{n+N} - \frac{1}{n+N} \cdot \lfloor \frac{n}{T} \rfloor \sum_{r=0}^{T-1} w_{k+N} \right| &\leqslant \frac{N+T-1}{n+N} \|\mathcal{W}\|_\infty
\end{aligned}$$

Now it can be proven that:

$$\lim_{n\to+\infty} \frac{1}{n+N} \cdot \lfloor \frac{n}{T} \rfloor \sum_{r=0}^{T} w_{k+N} = \frac{1}{T} \sum_{r=0}^{T-1} w_{r+N}$$

With that:

$$\lim_{n\to+\infty} \frac{1}{n} \sum_{k=0}^{n-1} w_k = \frac{1}{T} \sum_{r=0}^{T-1} w_{r+N} \quad \blacksquare$$

Now, our algorithm will be composed of 3 main parts:

- Calculating the transition function $F : V \times P \to V \times P$. This is straightforward from the construction.
- Calculating the period and the offset of the sequence. We will use Floyd's cycle finding algorithm for that.
- Calculating the Mean Payoff

This is an illustrative implementation of our algorithm.

Rami ZOUARI
INSAT, Tunisia

Algorithm 1 Deterministic strategies evaluation

Require: $G = (V, E, P, s, p)$ a mean payoff game
Require: $(\Phi^{\text{Max}}, \Phi^{\text{Min}})$ the edge probability
Ensure: R The mean payoff
  $F \leftarrow \text{Transition}(G, \Phi^{\text{Max}}, \Phi^{\text{Min}})$ ▷ Calculate the transition function
  $x_0 \leftarrow (s, p)$
  $(T, r) \leftarrow \text{FloydCycleFinding}(F, x_0)$ ▷ Find the period and the offset
  $S \leftarrow 0$ ▷ S represents the cumulative payoffs along a cycle
  $x \leftarrow x_0$
  for $k \in \{1, \dots, r\}$ **do** ▷ Advance until arriving to the cycle
    $x \leftarrow F(x)$
  end for
  for $k \in \{1, \dots, T\}$ **do**
    $y \leftarrow\!\leftarrow F(x)$
    $u \leftarrow \underset{V \times P \to V}{\text{projection}}(x)$ ▷ Extracts the current vertex
    $v \leftarrow \underset{V \times P \to V}{\text{projection}}(y)$ ▷ Extracts the next vertex
    $S \leftarrow S + W(u, v)$
    $x \leftarrow y$
  end for
  return $R \leftarrow \frac{S}{T}$

2.3.3 Probabilistic Strategies

Due to the undeterministic nature of probabilistic strategies, it does not make sense to evaluate the mean payoffs, as different executions may lead to different mean payoffs.
Instead, probabilistic strategies gives rise to a discrete distribution of mean payoffs.
Now two closely related, but different evaluations are possible

- Expected Mean Payoff
- Distribution of winners

Now, with both strategies fixed. A Mean Payoff Game can be considered as a Markov Chain.

Algorithm 2 Probabilistic strategies evaluation

Require: $G = (V, E, P, s, p)$ a mean payoff game
Require: $(\Pi^{\text{Max}}, \Pi^{\text{Min}})$ the edge probability
Ensure: $\mathbb{E}[R]$ The expected mean payoff
  $(A, W) \leftarrow \text{MRP}(G, \Phi^{\text{Max}}, \Phi^{\text{Min}})$ ▷ Extract the MRP form. This is detailed in C.1.2
  $u \leftarrow (A \odot W)\,\mathbf{1}$
  $X \leftarrow \text{NullSpace}(\text{Id} - A)$ ▷ Extract the kernel-basis of $\text{Id} - A$
  $Y \leftarrow \text{NullSpace}(\text{Id} - A^T)$ ▷ Extract the kernel-basis of $\text{Id} - A^H$.
  $T \leftarrow X(Y^T X)^{-1} Y^T$ ▷ Calculate the limit $\lim_{n \to +\infty} \frac{1}{n} \sum_{k=0}^{n-1} A^k$. This is detailed in C.2.3
  return $\mathbb{E}[R] \leftarrow Tu$

Rami ZOUARI
INSAT, Tunisia

2.4 Countering Strategies

By fixing the strategy of player pP to Φ^p, then the Markov Game Process reduces to a Markov Decision Process, which can be solve by Linear Programming tools
Moreover, in a deterministic Mean Payoffs, a counter strategy can be calculated efficiently by reducing the problem to finding a negative cycle in a di-graph.

2.4.1 Deterministic Counter Strategy

For simplicity, we assume that $p = \text{Max}$, the same results apply for Min player.
For a Mean Payoff Game (V, E, P, s, p'), we introduce the following graph:

$$G' = \big(V, E', W'\big)$$

where:

- E' is defined as follow:
$$E' = \{(u, \Phi^p(v)), \quad (u, v) \in E\}$$
- Also, W' is defined adequately:
$$\forall (u, v) \in E : W'(u, \Phi^p(v)) = W(u, v) + W(v, \Phi^p(v))$$

The problem of finding a counter strategy will be reduced to finding a negative cycle $\mathcal{C}$ in G'
This can be done in $\mathcal{O}(|V|^3)$

2.4.2 Probabilistic Counter Strategy

On the other hand, if Φ^p is probabilistic, then the problem can be reduced to optimizing the mean payoff of an infinite-horizon Markov Decision Process.

2.5 Learning Strategies

2.6 State of the Art

Mean Payoff Games are well-known in many fields, such that Optimization[2], Game Theory[11], Formal Verification[14], Constraint Satisfaction Problems[5, 3], Reinforcement Learning[35].
While we were not able to trace the exact origin of Mean Payoff Games, we were able to find references to it since the seventies [11]. The problem itself is interesting as it connect many related fields. First of all, it is closely related to many problems in constraint satisfaction [5, 3], model-checking [14], game theory [11].
Also, another interesting fact is that deciding the winner of a mean payoff game is polynomial time equivalent[10] to the Max Atom problem [3], which is in $\texttt{NP} \cap \texttt{co-NP}$, but its membership to $\texttt{P}$ is still open. This is remarkable, there only few problems that share such fate [29].

This influenced mainly two research axes. The first deals with solving the decision problem[11], and also optimization problem[12] related to calculating the optimal strategies.

[10] Each instance of both problems can be transformed to the latter in polynomial time.
[11] The decision problem of a mean payoff game is deciding the winner.
[12] The optimization problem is calculating the best strategy for each player.

The optimization problem itself can be solved using exact methods [3], as well as iterative methods [35, 2].

While we did not find a machine learning approach on mean payoff games in the literature, we were able to find some results in a superclass, known as stochastic parity games. In fact, a model free reinforcement learning approach was proposed using the Q-learning minimax algorithm , as well as a supervised learning approach on solved instances of that game.

2.7 Library Implementation

We have two implementations of our mean payoff game library:

- The first is called **mpg**. It is implemented in Python, and it contains the core functionalities of mean payoff games, as well as visualisation and support for machine learning methods.
- The second one is called **mpgcpp**. It is implemented in C++ to maximimize effeciency.

Also, the time-critical C++ functions are exported to Python. In they can be used via the **wrapper** module.

2.7.1 mpg

We have implemented a library called **mpg** that contains all the functionalities that we have discussed for mean payoff games.
Here we list the modules of that library:

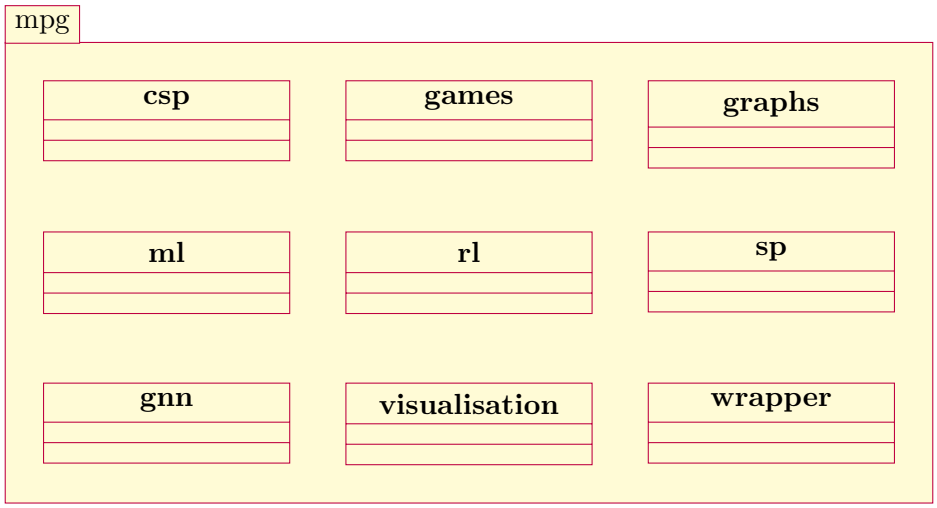


Figure 2.2: **mpg** library

This library contains the following modules:

csp

This module contains the constraint satisfaction methods used to solve Mean Payoff Games. They are describe in details in the appendix A.

graphs

This modules contains some graph algorithms needed for Mean Payoff Games, such as:

- Floyd-Warshall method to find negative cycles.
- Methods to generate random graphs as described in section 3. The theoretical details are in the appendix B

games

This module defines the core functionalities related to mean payoff games.

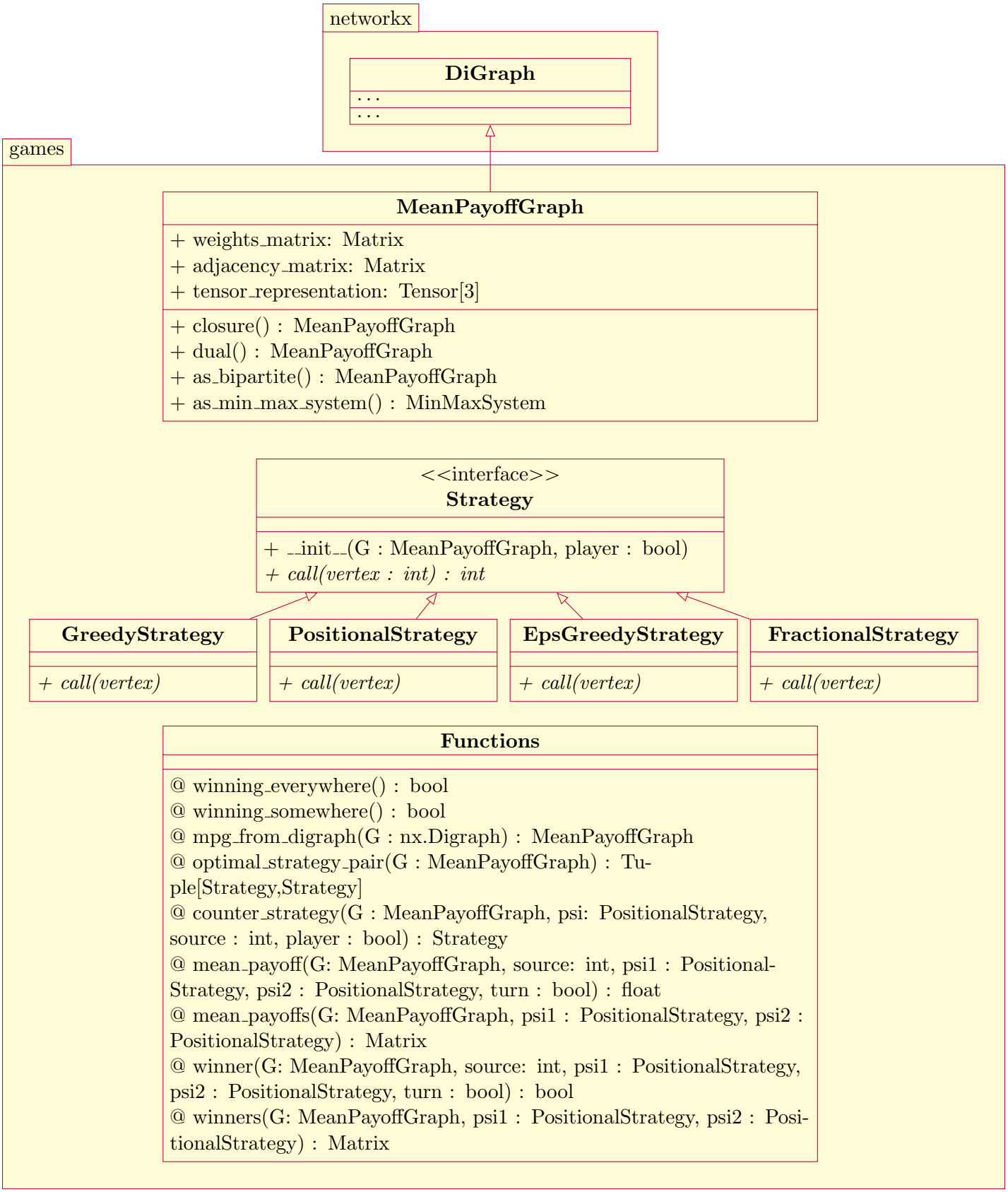


Figure 2.3: **games** module

visualisation

This module serves as a front-end for Jupyter Notebook, so we can visualise mean payoff games, and also the strategy of each player.

ml

This module defines the required layers, blocks, and model architectures to do machine learning on Mean Payoff Games. This is detailed in chapter 4.

gnn

This module defines the basic functionalities of graph neural networks [34] that are required for our models.

rl

This module defines the required functions to do reinforcement learning on Mean Payoff Games[13].

sp

This module defines a basic AlphaZero based agent to learn the game.

wrapper

This module contains a binding to the C++ implementation of time-critical methods for mean payoff games.
We will give details about this wrapper in the next section.

2.7.2 mpgcpp

This library contains the

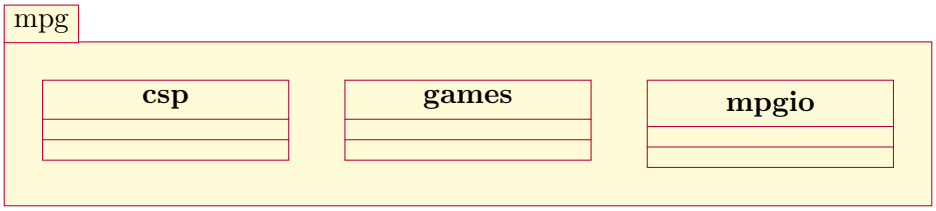


Figure 2.4: **mpgcpp** library

csp

This module contains the constraint satisfaction methods used to solve Mean Payoff Games.

[13]The current version of this module only supports Mean Payoff Games where at least one player has a fixed strategy.

games

This module defines the core functionalities related to mean payoff games.

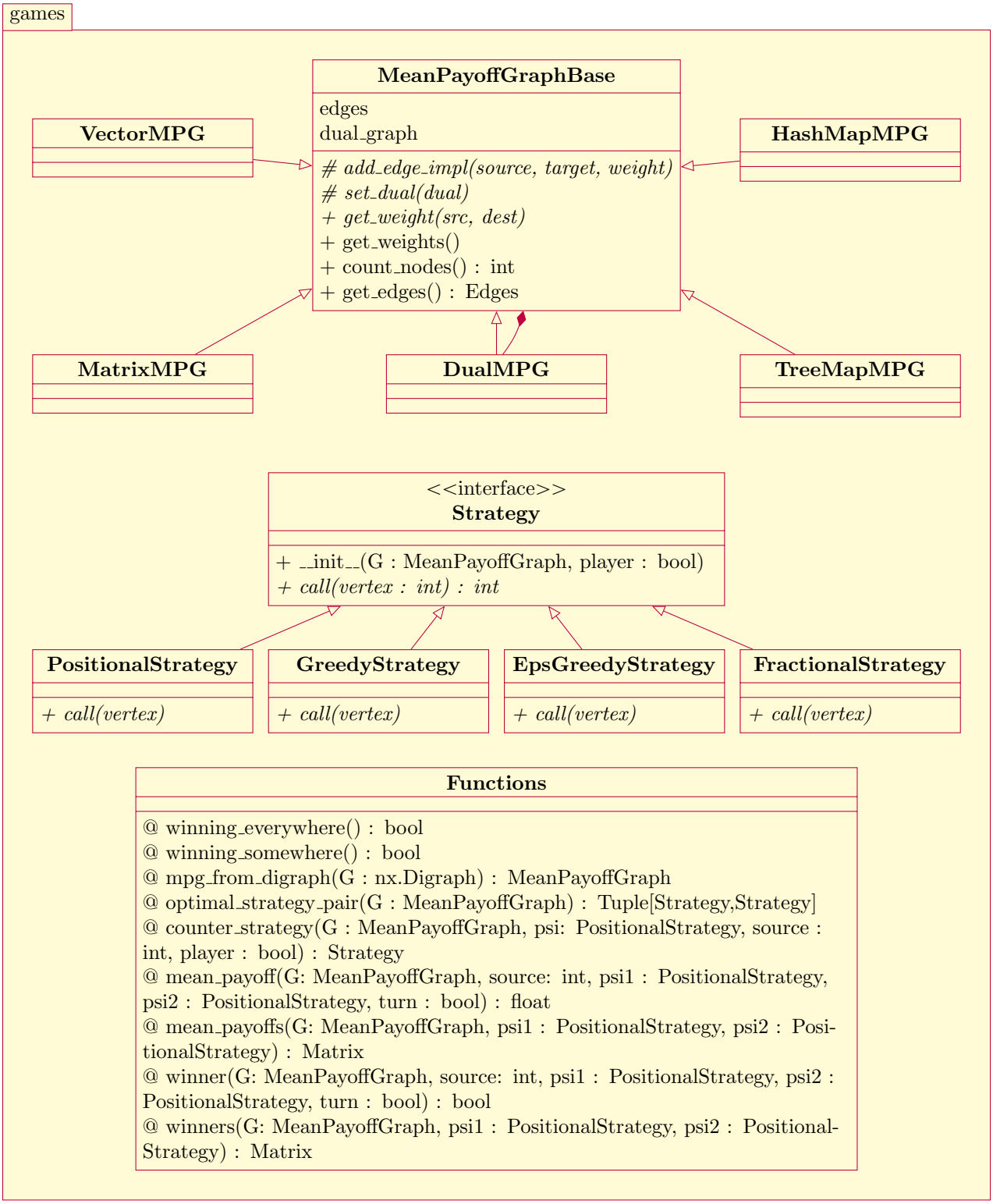


Figure 2.5: **games** module

2.7.3 Environment

2.7.4 Structure

Chapter 3

Dataset Generation

3.1 Introduction

3.2 Analysis

Generating a Mean Payoff Game can be decomposed into two subsequent objectives.

1. Generate the Graph itself.
2. Generate the Weights

3.3 Graph Distributions

There are many well studied graph distributions in the litterature.
One of the most explored ones are the $\mathcal{G}(n,p)$ and $\mathcal{G}(n,m)$ families.

3.3.1 $\mathcal{G}(n,p)$ Family

For $n \in \mathbb{N}, p \in [0,1]$, a graph G is said to follow a $\mathcal{G}(n,p)$ distribution if $|V| = n$ and:

$$\forall e \in \mathscr{E}, \quad \mathscr{P}(s \in \mathcal{E}) = p$$

Where $\mathscr{E}$ is a set of valid edges.

3.3.2 $\mathcal{G}(n,m)$ Family

For $n \in \mathbb{N}, m \in \mathbb{N}$, a graph G is said to follow a $\mathcal{G}(n,m)$ distribution if $|V| = n, |E| = m$ and the edges $e_1, \ldots, e_m$ were drawn from a set of valid edges $\mathscr{E}$.

3.3.3 Valid edges

The set of valid edges $\mathscr{E}$ is the set defining the potential edges of the graph. It is equal to:

1. $V \times V$ for directed graphs with loops
2. $(V \times V) \backslash V \odot V$ for directed graphs without loops
3. The set of subsets of size 2 of V denoted $\mathscr{P}_2(V)$ for undirected graphs with loops.
4. The set of subsets of size 2 of V denoted $\mathscr{P}_2(V)$ for undirected graphs with loops.

3.3.4 $\mathcal{D}(n,p)$ Graph Construction

Naive Method

The definition of $\mathcal{D}(n,p)$ gives a straightforward construction.
This is achieved by flipping a coin[1] for each pair of node $(u,v) \in V^2$, we add an edge if we get a Head.
This is implemented in the following algorithm:

Algorithm 3 $\mathcal{D}(n,p)$ Graph Generation

Require: $n \in \mathbb{N}^*$ the size of the graph
Require: $p \in \mathbb{N}^*$ the edge probability
Ensure: $G \sim \mathcal{D}(n,p)$
  $A : (u,v) \in V \times V \to 0$
  for $u \in V$ **do**
    for $v \in V$ **do**
      Generate $X \sim \mathcal{B}(p)$ ▷ $\mathcal{B}(p)$ is the bernoulli distribution
      $A(u,v) \leftarrow X$
    end for
  end for
  return $G \leftarrow$ `GraphFromAdjacencyMatrix`(A)

The complexity[2] of the following algorithm is $\mathcal{O}(n^2)$.

Optimized Method

Instead of iterating over all possible pair of nodes. For each vertex $v \in V$:

- We can sample a number d from the outgoing degree distribution[3]
- We then choose d numbers uniformly without replacement from an indexable representation of V

The following algorithm implements the optimized method:

Algorithm 4 $\mathcal{D}(n,p)$ Graph Generation Optimisation

Require: $n \in \mathbb{N}^*$ the size of the graph
Require: $p \in \mathbb{N}^*$ the edge probability
Ensure: $G \sim \mathcal{D}(n,p)$
  $A : u \in V \to \varnothing$
  for $u \in V$ **do**
    Generate $d \sim \mathcal{B}(n,p)$ ▷ d represents the degree, $\mathcal{B}(n,p)$ is the binomial distribution
    $A(u) \leftarrow \text{choice}(V,d)$
  end for
  return $G \leftarrow$ `GraphFromAdjacencyList`(A)

[1] The coin is potentially biased with a probability of obtaining head equal to $p \in [0,1]$
[2] We assume the cost of generating a Bernoulli random variable as $\mathcal{O}(1)$
[3] Or the ingoing degree distribution, they are in fact equal.

Now, let $C(a, b)$ be the cost of choice function.
The expected complexity of this algorithm as a function of $n = |V|$ and the degrees $d_1, \ldots, d_n$ is:

$$\tilde{\mathcal{O}}\left(\sum_{i=1}^{n} 1 + \mathbb{E}[C(n, d_i)]\right) = \tilde{\mathcal{O}}\left(n + \sum_{i=1}^{n} \mathbb{E}[C(n, d_i)]\right) \tag{3.1}$$

We will show on the next section what choice function should we use.

3.3.5 Choice Function

First Proposition

We propose here a simple choice algorithm, but it is still efficient for our use case.
It works simply by drawing without replacement, but we ignore duplicate elements. This is implemented as follow

Algorithm 5 $\mathcal{D}(n, p)$ Choice without replacement

Require: S a list
Require: $m \in \{0, \ldots |S|\}$ the number of chosen elements
Ensure: H a set of size m containing uniformly drawn elements without replacement.
  $H \leftarrow \varnothing$
  while $|H| < m$ **do**
    Generate $v \sim \mathcal{U}(S)$ ▷ Where $\mathcal{U}(S)$ is the uniform distribution over S
    $H \leftarrow H \cup \{v\}$
  end while
  return H

To estimate the cost of this algorithm, we will use probabilistic reasoning.
Let $X_{n,m} = C(n, m)$ the running time of an execution of algorithm 5 in a set S of size n, with m elements to be chosen. We have:

$$\begin{aligned}
&X_{n,0} \text{ is deterministic} \\
&X_{n,0} = \mathcal{O}(1) \\
\mathbb{E}[X_{n,m}] &= 1 + \frac{1}{n}\sum_{k=0}^{n-1} \mathbb{E}[X_{n,m} \mid \text{The last drawn number is } k] \\
&= 1 + \frac{1}{n}\sum_{k=0}^{m-2} \mathbb{E}[X_{n,m}] + \frac{1}{n}\sum_{k=m-1}^{n-1} \mathbb{E}[X_{n,m-1}] \\
&= 1 + \frac{m-1}{n}\mathbb{E}[X_{n,m}] + \frac{n-m+1}{n}\mathbb{E}[X_{n,m-1}]
\end{aligned}$$

Rami ZOUARI
INSAT, Tunisia

Now we arrived at a recurrent formula. We will simplify it as shown below:

$$\begin{aligned}
\frac{n-m+1}{n}\mathbb{E}[X_{n,m}] &= \frac{n-m+1}{n}\mathbb{E}[X_{n,m-1}] + 1 \\
\implies \mathbb{E}[X_{n,m}] &= \frac{n-m+1}{n-m+1}\mathbb{E}[X_{n,m-1}] + \frac{n}{n-m+1} \\
&= \mathbb{E}[X_{n,m-1}] + \frac{n}{n-m+1} \\
&= \sum_{k=1}^{m} \frac{n}{n-k+1} + \mathcal{O}(1) \\
&= \sum_{k=0}^{m-1} \frac{n}{n-k} + \mathcal{O}(1) \\
&= n \sum_{k=n-m+1}^{n} \frac{1}{k} + \mathcal{O}(1) \\
&= n(H_n - H_{n-m}) + \mathcal{O}(1)
\end{aligned}$$

Here $(H)_{n\in\mathbb{N}*}$ is the harmonic series, and we define $H_0 = 0$.

Complexity

The expected complexity of algorithm 5 depends on both n and m:

- If $m = kn + o(n)$ with $k \in]0, 1[$, then it is $\tilde{\mathcal{O}}(m)$.
- If $m = n - o(n)$, It is[4] $\tilde{\mathcal{O}}(m \log m)$.

To prove this result, we use a well-known asymptotic approximation of the Harmonic series[5] [21, Section 1.2.11.2]:

$$H_n = \ln n + \gamma - \frac{1}{2n} + \mathcal{O}\left(\frac{1}{n^2}\right)$$

We can prove this claim as follow for $m = km + o(m), k \in [0, 1[$:

$$\begin{aligned}
\mathbb{E}[C(n,m)] &= -n \ln\left(1 - \frac{m}{n}\right) - \frac{1}{2}\left(1 - \frac{n}{n-m}\right) + \mathcal{O}\left(\frac{1}{n}\right) \\
&= -n \ln(1 - k + o(1)) + \frac{1}{n}(1 - \tfrac{1}{1-k+o(1)}) + \mathcal{O}(\tfrac{1}{n}) \\
&= \mathcal{O}(m)
\end{aligned} \tag{3.2}$$

For $m = n - o(n)$, we prove it by noting that:

$$\mathbb{E}[C(n,m)] \leqslant \mathbb{E}[C(n,n)] = \mathcal{O}(nH_n) = \mathcal{O}(m \log m)$$

Refinement

If m tends to n, it is more hard to select m elements from a set of size n without replacement. This explains the extra logarithmic factor.

[4] Here we use the minus sign to emphasize that $m \leqslant n$

[5] This asymptotic approximation can be proven using the Euler–Maclaurin formula

In that case, we can instead focus on the dual problem: "Find the $n-m$ elements that will not be selected". This can be calculated in $\mathcal{O}(n-m)$.
Once we find the elements that will not be selected, their set complement are exactly the m elements that will be selected. This new algorithm is guaranteed to be $\mathcal{O}(m)$ irrespective of n and m

Algorithm 6 Fine tuned $\mathcal{D}(n,p)$ Choice without replacement

Require: S a list
Require: $m \in \{0, \dots |S|\}$ the number of chosen elements
Require: choice The choice function defined on algorithm 5
Require: τ a fine tuned threshold. We will use $\tau = \frac{1}{2}$ for all practical purposes.
Ensure: H a set of size m containing uniformly drawn elements without replacement.
  if $\frac{m}{|S|} \leqslant \tau$ **then**
    $H \leftarrow \text{choice}(V, n)$
  else
    $H \leftarrow S \backslash \text{choice}(S, n-m)$
  end if
  return H

Also, an important point is that by combining the analysis of all possible cases, we can extract a constant factor that is independent[6] of n. So that the Big-O bound is only a dependent on m.

3.3.6 Complexity of Optimised $\mathcal{D}(n,p)$ Graph Construction

Now, using algorithm 6 as the choice algorithm, we can further simplify equation (3.1) as a function of only $n = |V|$ and $m = |E|$.
This proves that $\mathcal{D}(n,p)$ construction can be achieved in expected linear time:

$$\begin{aligned}\tilde{\mathcal{O}}\left(n + \sum_{i=1}^{n} \mathbb{E}[C(n,d_i)]\right) &= \tilde{\mathcal{O}}\left(n + \sum_{i=1}^{n} d_i\right) \\ &= \tilde{\mathcal{O}}\left(n+m\right)\end{aligned}$$

3.3.7 $\mathcal{D}(n,m)$ Construction

To construct a random $\mathcal{D}(n,m)$ graph, we only have to select m uniformly random elements from the set $V \times V$.
We will use algorithm 6 for this purpose[7]:

Algorithm 7 Fine tuned $\mathcal{D}(n,p)$ Choice without replacement

Require: $n \in \mathbb{N}^*$
Require: $m \in \{0, \dots, n^2\}$ the number of chosen elements
Ensure: $G \sim \mathcal{D}(n,m)$
  $E \leftarrow \text{choice}(\text{Lazy}(V) \times \text{Lazy}(V), m)$ ▷ We only need the m elements on-demand.
  return $G \leftarrow \text{GraphFromEdges}(E)$ ▷ This justifies using Lazy

[6] The independence can be proven by taking the supremum of the right-hand side of equation (3.2) over $[0, \tau]$, and the fact that τ is fixed.

[7] It is essential that the list $V \times V$ be lazy loaded. In particular, each element will only be loaded when it is indexed. This is essential to reduce the complexity. Otherwise, we will be stuck in an $\mathcal{O}(n^2)$ algorithm.

Here Lazy(V) $\times$ Lazy(V) is a lazy implementation of cartesian product that supports bijective indexing[8] over $\{0, \dots, n^2 - 1\}$.
The complexity of this construction is: $\tilde{\mathcal{O}}(m)$

3.4 Sinkless Conditionning

Sampling from a graph distribution may lead to graphs that have at least one sink.
These graphs are problematic as Mean Payoff Graphs are exactly the sinkless graphs.
To migitate this, we will impose a conditionning on both distribution that will gives a guaranteed Mean Payoff Graph.
We will explore such conditionning both distribution:

- $\mathcal{G}^S(n, p)$: This is the distribution of graphs following $\mathcal{G}(n, p)$ with the requirement that they do not have a sink.
- $\mathcal{G}^S(n, m)$: This is the distribution of graphs following $\mathcal{G}(n, m)$ with the requirement that they do not have a sink.

3.4.1 Repeating Construction

Algorithm

This method is very intuitive. It will repeat the sampling until getting the desired graph.
The following is an implemention of the repeating construction.

Algorithm 8 Fine tuned $\mathcal{D}(n, p)$ Choice without replacement

Require: $n \in \mathbb{N}^*$
Require: $m \in \{0, \dots |S|\}$ the number of chosen elements
Require: choice The choice function defined on algorithm 5
Require: Threshold τ
Ensure: H a set of size m containing uniformly drawn elements without replacement.
  if $\frac{m}{|S|} \leqslant \tau$ **then**
    $H \leftarrow \text{choice}(V, n)$
  else
    $H \leftarrow V \setminus \text{choice}(S, n - m)$
  end if
  return H

Analysis

We will analyse the runtime of generating a $\mathcal{G}^S(n, p)$.
We expect a similar runtime for $\mathcal{G}^S(n, m)$ due to the similarity between $\mathcal{G}(n, m)$ and $\mathcal{G}(n, p)$.
Let $F(n)$

[8] Indexing is required for uniform sampling

3.5 Weights Distribution

3.5.1 Construction

Once the graph is constructed. We only have to generate the weights.
This will be done by creating a random weight function:

$$W(u,v) : (u,v) \to W_{u,v}$$

Here $W_{u,v}$ will be a sequence of real random variables.
In our case, we set $(W_{u,v})_{(u,v)\in E}$ to be independent and identically distributed over a real distribution $\mathcal{W}$.

3.6 Proposed MPG Distribution

3.6.1 Desired Properties of Mean Payoff Game Distributions

Fairness in the Limit

This is essential, as we intend to generate a sequence of Mean Payoff Games that do not favour statistically a certain player, in the sense that, if we generate sufficient independent and identically distributed Mean Payoff Games $G_1, \ldots, G_n$, we expect the following:

$$\lim_{n\to+\infty} |\mathtt{R}_{\mathrm{Max}}(G_1, \ldots, G_n) - \mathtt{R}_{\mathrm{Min}}(G_1, \ldots, G_n)| = 0$$

Where $\mathtt{R}$ is defined as follow:

$$\mathtt{R}_{\mathrm{Op}}(G_1, \ldots, G_n) = \frac{1}{n}\sum_{i=1}^{n} \mathscr{P}(\text{Op wins } G_i \text{ assuming optimal strategies})$$

Symmetric

A real distribution is said to be symmetric if:

$$\forall [a,b] \in \mathbb{R}, X \sim \mathcal{W}, \quad \mathscr{P}(X \in [a,b]) = \mathscr{P}(X \in [-b,-a])$$

We will define a symmetric Mean Payoff Game distribution as a distribution of Mean Payoff Game whose weights are independent and identically distributed on a symmetric real distribution. This property is stronger than Fairness in the Limit, as it implies that:

$$\mathscr{P}(\text{Max wins } G \text{ assuming optimal strategies}) = \mathscr{P}(\text{Min wins } G \text{ assuming optimal strategies})$$

We will require a Symmetric Mean Payoff Game as we do not want a player to have an inherit advantage other the other one[9]

3.6.2 Implemented Distributions

The following table resumes the implemented distributions:

[9] Other than the first move.

Distribution Family	Parameters	Type
$\mathcal{D}(n,p)$	• n : Graph size • p : Edge probability	Graph distribution
$\mathcal{D}(n,m)$	• n : Graph size • m : Number of edges	Graph distrbiution
$\mathcal{U}_{\text{discrete}}(-r,r)$	• r : The radius of the support	Weight distribution
$\mathcal{U}(-r,r)$	• r : The radius of the support	Weight distribution
$\mathcal{N}(0,\sigma)$	• σ : The standard deviation	Weight distribution

Table 3.1: Le tableau d'avancement des BNNs

Also, to generate the initial state, we have defaulted to:

- The uniform distribution over the vertices to generate the starting vertex
- The bernoulli distribution to generate the starting player.

With all that said, distributions are fair in the limit and are symmetric, provided that the underlying graph distribution and weight distribution are chosen from the table 3.1.

3.7 MPG Generation

3.7.1 Distribution

- Each generated graph will follow a distribution $\mathcal{G}(n, p(n))$ for some $n \in \mathbb{N}^*$
- The weights will follow the discrete uniform distribution $\mathcal{D}(-1000, 1000)$

We will generate two kinds of datasets, depending on the nature of the graph

3.7.2 Dense Graphs

- Let $\mathcal{P} = \{0.1, 0.2, 0.3, 0.5, 0.7, 0.8, 0.9, 1\}$
- $\mathcal{N} = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 200, 250, 300, 400, 500\}$
- For each $(n, p) \in \mathcal{N} \times \mathcal{P}$, we will generate $K = 1000$ observations $G_1^{n,p}, \ldots, G_K^{n,p} \sim \mathcal{G}(n, p)$

The total number of examples is:

$$K \times |\mathcal{N}| \times |\mathcal{P}| = 160000$$

The generation was done on a 'haswell64' partition with 24 cores. and it took 02:12:38 hours.

3.8 Annotation

3.8.1 Approach

We used the CSP algorithm 14 to annotate the dataset, potentially augmented with some heuristics. We implemented a program that takes the path of the dataset, and solves the Mean Payoff Games one by one.
To maximize efficiency, the program launches many solver threads, with each one independently working on a single file, and the results are accumulated using a ConcurrentQueue.

3.8.2 Target Values

The solver will calculate the following targets:

- The optimal pair of strategies
- The mean payoffs for each starting position, turn.
- The winners for each starting position, turn.

Also, some additional metadata are generated for analysis:

- `dataset`: The name of the whole dataset
- `graph`: The name of the graph.
- `status`: The solver's status on the given graph. In particular, whether it succeeded to solve the instance or not[10]. Equal to "OK" if the execution is successful.
- `running_time`: The time needed to solve the instance.

3.8.3 Heuristics

To accelerate the annotation of the two datasets, we had to apply some heuristics to the algorithm. We made essentially two kinds of heuristics.

Linear Bound Heuristic

This is the heuristic based on the view that for almost all solutions of a Ternary Max Atom system extracted from our generated random games, either:

- All variables are infinite:
$$X(u) = -\infty \quad \forall u \in V$$
- The diameter of assignments is in the order of $\|W\|_\infty$
$$\Delta X = \sup_{u \in V} X(u) - \inf_{u \in V, X(u) > -\infty} X(u) = \mathcal{O}(\|W\|_\infty)$$

[10] We expect that the solver may crash due to several reasons (corrupted file, out of memory, etc...). For that we made additional effort for exception handling, so that an error for a single instance does not propagate to the whole program.

This heuristic suggests a much tighter search space to the worst case $\|W\|_1$ one. Going further, with uniform random weights:

$$\|W\|_1 = \mathcal{O}\left(|E| \times \|W\|_\infty\right)$$

We believe this heuristic arises due to the random property of graphs, because in general, one can build an infinite family of ternary max atom systems that violate this heuristic.
In fact, going further, one can build ternary max atom systems were the $\|W\|_1$ estimation is tight.

To generating the dataset, we applied this heuristic with $\Delta X = 4\|W\|_\infty$

$$D = \{-\infty, -2\|W\|_\infty, , -2\|W\|_\infty + 1, \ldots, 2\|W\|_\infty\}$$

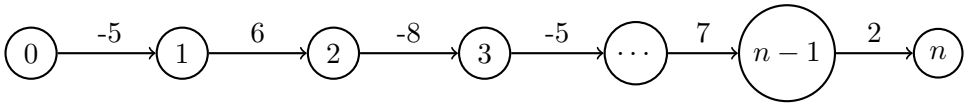


Figure 3.1: A counter example of the Linear Bound heuristic

Early Stopping

If after any iteration of arc consistency, $\max_{x \in V} \nu(x) < \sup D$. Then, $\nu(t)$ will converge to $-\infty$ for all t.
Thus, we stop the algorithm and sets $\nu(t) \leftarrow -\infty, \quad \forall t$

Preuve 1 *suppose that in fact there is an assignment with:*

$$-\infty < \max_{u \in V} \nu(u) < \sup D$$

We will take the u with the biggest such $\nu(u)$.
Now our system is a tropical max atom system, which means translations are also a polymorphism of this system, so for any assignment $\nu : V \to \mathbb{Z}$, $X + t$ is also an assignment $\forall t \in \mathbb{Z}$. With that, $\nu + \sup D - \nu(u)$ is also an assignment.
This assignment has the property:

$$\forall s \in V, \quad \nu(s) + \sup D - \nu(u) \in D$$

Which is a contradiction, as it violates the consistency of arc consistency, and the maximality of the solution with respect to the domain D

The efficiency of the Early Stopping heuristic depends on the density of the graph. Empirically, for dense graphs. the analoguous ternary max atom system usually has two kind of assignments:

1. Either all variables are finite
2. Either all variables are $-\infty$

This translates back in a dense Mean Payoff setting, that the winner of the game usually does not depend in the starting position and the starting turn.
With that, the Early Stopping heuristic will quickly detect the second case, which we believe as the hurdle of the algorithm.
On the other hand, for sparse graphs, we do not have this nice distinction between finite and infinite assignments, and they can overlap, and so will make this heuristic useless in practice.

3.8.4 Implementation

Algorithm

We implemented a Mean Payoff Graph solver. It calculates the optimal move for each player in each position. Thus, our implementation gives an exact solution to the optimization problem[11] for Mean Payoff Games.
It works by a transforming a mean payoff game to an equivalent min-max system, then applying two subsequent reductions to a n-ary max atom system, then to a ternary max atom system. The solution of the latter is propagated back to mean payoff game to induce an optimal strategy.

[11] In its current version, It gives a weak optimal strategy for both players Max, and Min . The winning condition is C_2

Algorithm 9 Solving a Mean Payoff Graph for all states

Require: G a Mean Payoff Graph
Require: D the domain of the variables. Can be chosen by heuristics.
Ensure: $\Phi : V \times P \to V$ The optimal strategy of each player

```
Φ ← ∅
for p ∈ P do
    if p is Max then
        G' ← G
    else
        G' ← Ḡ
    end if
    S ← transform_{MPG→Min-Max}(G', D)
    S' ← transform_{Min-Max→Max}(S)
    S'' ← transform_{Max→Max3}(S')
    L ← inf D
    R ← sup D
    Q ← ∅
    V ← Variables(S'')
    for u ∈ V do
        X(u) ← R
        append(Q, u)
    end for
    while Q ≠ ∅ do
        X ← arcconsistency(S'', X, Q, L)
    end while
    for C ∈ S'' do                                   ▷ Iterate over constraints of S''
        OP ← Operator(C)                             ▷ Get the operator of C. Either Max or Min
        Y the right-hand side variables of C
        C the right-hand side constants of C
        x the left-hand side variable of C
        u ← projection_{V×P→V}(x)                    ▷ Extract the vertex
        if OP is Max then
            y*, c* ← argmax_{(y,c)∈zip(Y,C)} {X(y) + c}   ▷ Extracts the maximum assignment
            Φ(u, p) ← projection_{V×P→V}(y*)          ▷ The Strategy is the vertex of the maximum assignment
        end if
    end for
end for
return Φ
```


Rami ZOUARI
INSAT, Tunisia

Here $\text{zip}(Y, C)$ of lists Y and C is the list $L = [(y_1, c_1), \ldots, (y_n, c_n)]$

These transformations were not trivial to find, we had to improve the reductions offered by [3], we also proposed a refinement to arc consistency that works for a ternary max-atom system, that takes advantage of polymorphisms, as well as the symmetries.
This is very technical, and for that, the details are listed in the appendix A.

Complexity Analysis

For simplicity will suppose that the domain D is finite[12], our algorithm runs in:

$$\mathcal{O}\left((|V| + |E|)^2 \cdot |D|\right)$$

Otherwise, if D is a real domain, the algorithm still converges if D is bounded[13], since arc consistency takes a function and produces a smaller one. With that said, we did not produce a complexity estimation, as our work directly relies to the generated graphs that we have discussed in section 3.7, we recall that their weights are finite.

3.8.5 Deployment

After some experiments, it was very clear that vertical scaling with the number of threads is not sufficient. By analysing the running time of some samples, we estimated the total running time solving both datasets to exceed 30 days.
As a result of this, we deployed a pipeline of 24 nodes, each with 24 threadss working simultaneously on a partition of the dataset.

[12] In particular, a finite subset of $\mathbb{Z}$
[13] $D \subseteq [a, b]$ for some interval $[a, b]$

Chapter 4

Model Design

4.1 Introduction

4.2 Objectives

The main objective

4.3 Properties

We expect the model $\mathcal{M}$ to verify the following properties:

4.3.1 Totality

Definition

A function is total if it is defined in the whole domain.
Also, we will say a model $\mathcal{M}$ is total if it works on all Mean Payoff Games..

Importance

This property implies two main characteristics:

1. The model works for any Mean Payoff Game whatever the size of $|V|$. This is tricky as most machine learning models act on a batch of data with a fixed shape.

2. The model works for any Mean Payoff Game whatever the representation is; This is implicitely verified by an encoding of the Mean Payoff Games.

4.3.2 Node Agnostic

This property tells that a model should not use any extra information about the nodes.

Formalisation

Let $G_1(V_1, E_1, W_1, s_1), G_2(V_2, E_2, W_2, s_2)$ two isomorphic Mean Payoff Games in the sense that there exists a bijection $\Phi : V_1 \to V_2$ such that:

$$\begin{aligned} E_2 &= \{(\Phi(u), \Phi(v)), \quad (u, v) \in E_1\} \\ \forall (u, v) \in E_1, \quad W_2(u, v) &= W_1(\Phi(u), \Phi(v)) \\ s_2 &= \Phi(s_1) \end{aligned}$$

Then:

$$\Phi\left(\mathcal{M}(G_1)\right) = \mathcal{M}(G_2)$$

Explanation

The property tells that if two graphs mean payoffs only differ by their node representation, then the results should also only differ by the representation of the nodes.

Importance

This property implies that we can simply encode a mean payoff as $G(V, E, W, s)$ as $G'(V', E', W', s')$ with $V' = \{0, ;|V| - 1\}$, and E', W', s' defined accordingly.
In fact, the encoding is done implicitely by our model[1].

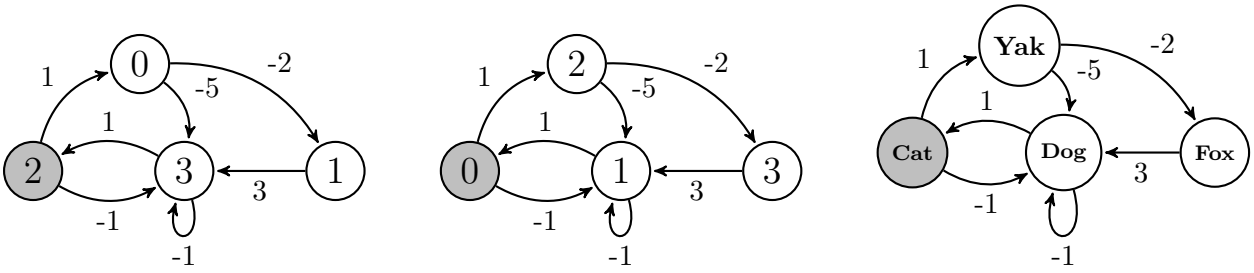


Figure 4.1: Three isomorphic Mean Payoff Games

In the figure 4.1, we have 3 equivalent Mean Payoff Games. To illustrate a node agnostic model:

- Let $\mathcal{M}$ be a model that predicts for a given Mean Payoff Game, the action to be played at the starting node, and the winner.
- Suppose also that $\mathcal{M}(G_1) = (3, \text{Max})$

If $\mathcal{M}$ is node agnostic, then:

$$\begin{aligned} \mathcal{M}(G_2) &= (1, \text{Max}) \\ \mathcal{M}(G_3) &= (\text{Dog}, \text{Max}) \end{aligned}$$

[1]While the encoding is done implicitely, the model itself can violate this property. An example of this is a Multi Layer Perceptron. Which lead to different results for different encodings.

4.3.3 Invariance under Positive Scaling

This property comes directly from the fact that both the winner and the set of optimal strategies[2] are invariant under a positive scaling of the weights.
Now, it is very easy to make augment a model $\mathcal{M}$ into such invariant model $\mathcal{M}'$. We only do the following:

$$\mathcal{M}'(E, W, s) = \mathcal{M}'(E, \text{Normalize}(W), s)$$

Where Normalize is any endomorphism of weight functions that verify the following constraint:

$$\forall_{\text{MPG}} G(E, W), \forall, \quad \text{Normalize}(W) = \frac{W}{H(W)}$$

With $H : \mathscr{F}(E, \mathbb{R}) \to \mathscr{F}(E, \mathbb{R})$ a function satisfying[3]:

$$\forall s \in \mathbb{R}_+^*, \quad H(sW) = sH(W)$$
$$H(W) > 0$$

Standard Scaling

This scaling treats the values of W as samples of random variables, and divides W by an estimate of their variance.

$$\text{Normalize}(W) = \frac{W}{\sqrt{\mathbb{V}[W]}}$$
$$\mathbb{V}[W] = \frac{1}{|E|} \sum_{(u,v)\in E} (W(u, v) - \mathbb{E}[W])^2$$
$$\mathbb{E}[W] = \frac{1}{|E|} \sum_{(u,v)\in E} W(u, v)$$

Maximum Scaling

This scaling reduces the interval of the weights to $[-1, 1]$ by dividing by the largest weight in terms of absolute value:

$$\text{Normalize}(W) = \frac{W}{\|W\|_\infty}$$

Implementation Notes: The weights function is implemented as a matrix W, which is equal to 0 for $(u, v) \notin E$.
It is important to igonre these zeros[4] in both normalisations, otherwise it may lead to biased scaling.

[2] Whatever the definition of optimality (Weak, Strong, Payoff).
[3] Special care must be when H is $\geqslant$ instead of $>$
[4] And only these zeros. If $W(u, v) = 0$ for $(u, v) \in E$, this term should be accounted.

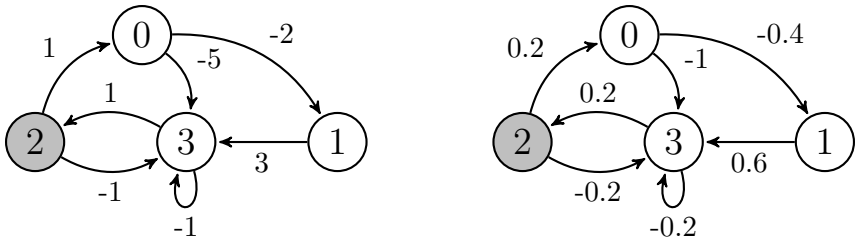


Figure 4.2: Mean Payoff Game, with a rescaled version using Maximum normalization

4.3.4 Permutation Equivariance

Definition

This property states that the model should ouptut the same results for permuted nodes, up to permutation of the results.

Importance

While it is a special case of the Node Agnostic property, but it is still important as the encoding of the graphs is done implicitely, and thus Permutation Equivariance is enough to get a Node Agnostic model.

4.3.5 Stability under Padding

Definition

- Let $G_1(V_1, E_1, W_1, s_1), G_2(V_2, E_2, W_2, s_2)$ be two disjoint Mean Payoff Games, in the sense that $V_1 \cap V_2 = \varnothing$.
- Let $B = (E_3, W_3)$ with $E_3 \subseteq V_2 \times V_1$ a bridge from G_2 to G_1, and W_3 is the weight function of E_3

The padding of G_1 by G_2 using B as a bridge, denoted as $G_1 \overset{B}{\triangleright} G_2$ is defined as:

$$G_1 \overset{B}{\triangleright} G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup E_3, W_1 \cup W_2 \cup W_3, s_1)$$

If the bridge B is empty, in the sense that $E_3 = \varnothing$, then we denote we will simplify the notation to $G_1 \triangleright G_2$.
A model is said to be stable under padding if:

$$\mathcal{M}(G_1 \overset{B}{\triangleright} G_2) = \mathcal{M}(G_1)$$

Importance

Deep Learning algorithms generally accept batches of data having a homogeneous shape.
In the other hand, graph input generally has different shapes, and thus are problematic to most learning algorithms.
While we succeeded in experimenting a learning algorithm with a ragged batch[5], it suffered the following limits:

[5] A batch of inhomogeneous input

- It greatly limits the choice of potential models.
- The training is not supported by GPU, as some core operations were only implemented in the CPU for ragged batches.

For this reasons, we opted to pad the graphs to get homogeneous batches, and this is why the stability under padding is important.

Removing Unreachable Nodes

Another major point for this property, is it gives the possibility to remove unreachable nodes without affecting the model's results.

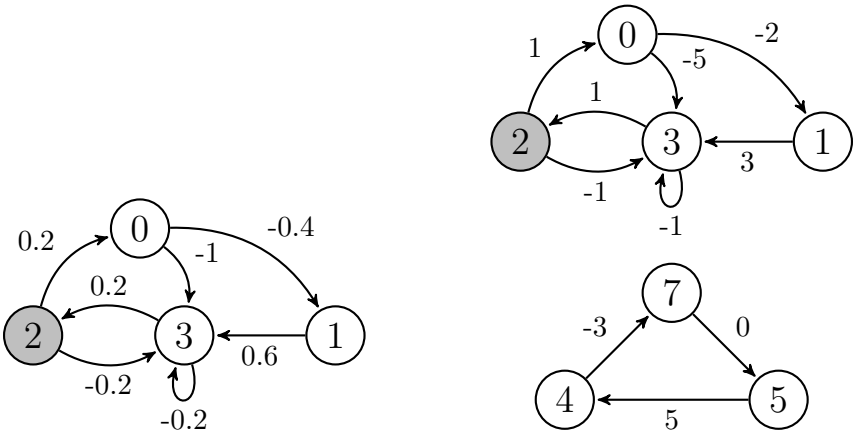


Figure 4.3: Original Mean Payoff Game with a padded version

Following the example under figure 4.3, if the model $\mathcal{M}$ is stable under padding, then the triangular subgraph should not be accounted in the output.

4.4 Considered Models

While there are many possible predictive models. We considered mainly two families of predictive models.

4.4.1 Value based Model

Such model predicts the evaluation of a certain position.
The evaluation is a function $\mathcal{M} : V \times P \to [-1, 1]$ with the following interpretation:

- $\mathcal{M}(s, p) = 1$ when the model predicts that player p will win the game given the position.
- $\mathcal{M}(s, p) = -1$ whens the model predicts that player p will lose the game given the position.
- $\mathcal{M}(s, p) = 0$ when the model predicts that the position will result in a draw.

Now, while such a model can be used to predict the winner. We claim that it is not efficient[6] to extract the strategy from such model, as the information of winning alone does not directly induce a strategy.
The following figure illustrates an example.

[6]Efficient in the sense that we cannot extract the best strategy from the evaluation in linear time.

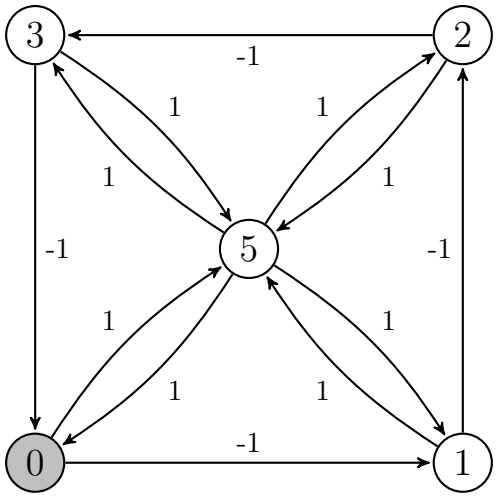


Figure 4.4: An MPG where the first player always wins.

4.4.2 Strategy based Model

Such model predicts the strategy for each player.
This is a function $\mathcal{M} : V \times P \to \mathscr{P}(V)$ such that:

$$\begin{aligned} \forall (u,p) \in V \times P, \quad & \mathcal{M}(u,p) \in \operatorname{Adj} u \\ \forall (u,p) \in V \times P, \forall v \in \operatorname{Adj} u, \quad & \mathcal{P}\left(\mathcal{M}(u,p) = v\right) \in [0,1] \\ \forall (u,p) \in V \times P, \quad & \sum_{v \in \operatorname{Adj} u} \mathcal{P}\left(\mathcal{M}(u,p) = v\right) = 1 \end{aligned}$$

4.5 Building the Model

Our model follows from well esta../Figures/PreprocessingBlock.pngblished graph neural network architectures, with minor modifications to suit our needs.
It is composed of blocks, each containing a **Weighted Graph Convolutional Network**.

4.5.1 Preprocessing Block

The preprocessing block is responsible to convert the graph to a format recognizable by the convolutional blocks.
It takes the following steps:

1. It takes a batch, add a padding to each graph so that the resulting batch is of homogenous shape.
2. It adds some random edges to each graph, with some low probability `random_connection_probability`.
3. It normalizes the weights.
4. It adds some noise to the weights.

Padding Layer

First of all, we define H_k be the graph of size k in which all the edges are loops with 0 as a weight. Each graph $G^{(i)} = (A^{(i)}, W^{(i)})$ in the batch has a size r_i.
Let $r = \max_i r_i$. The padding layer adds a potential padding to each graph in the following manner:

$$G^{(i)} \leftarrow G^{(i)} \rhd I_{r-r_i}$$

With that, the resulting batch will be of shape $(?, r, r, 2)$

Random Connections

This layer adds an edge with some probability p :

$$A^{(i)} \leftarrow A^{(i)} \vee B^{(i)}$$

Where $B^{(i)}$ is a matrix of shape (r, r) whose elements are sampled from the bernoulli distribution $\mathcal{B}(p)$.

Weights Normalisation

We want the model to be invariant under positive scaling. To achieve that, we implemented a weights normalization layer conforming to section 4.3.5

Weights Noise

For regularization, we implemented a weights noise layer, that adds a small additive noise to the weights, in the following manner:

$$W^{(i)} \leftarrow W^{(i)} + N^{(i)} \odot A^{(i)}$$

Where $N^{(i)}$ is the noise matrix. We implemented two kinds of noise:

- Uniform noise, where each element of $N^{(i)}$ follows $\mathcal{U}(-r, r)$
- Gaussian noise, where each element of $N^{(i)}$ follows $\mathcal{N}(0, \sigma)$

One Hot Encoding

The one hot encoding layer, as its name suggests, applies one hot encoding to the position s.

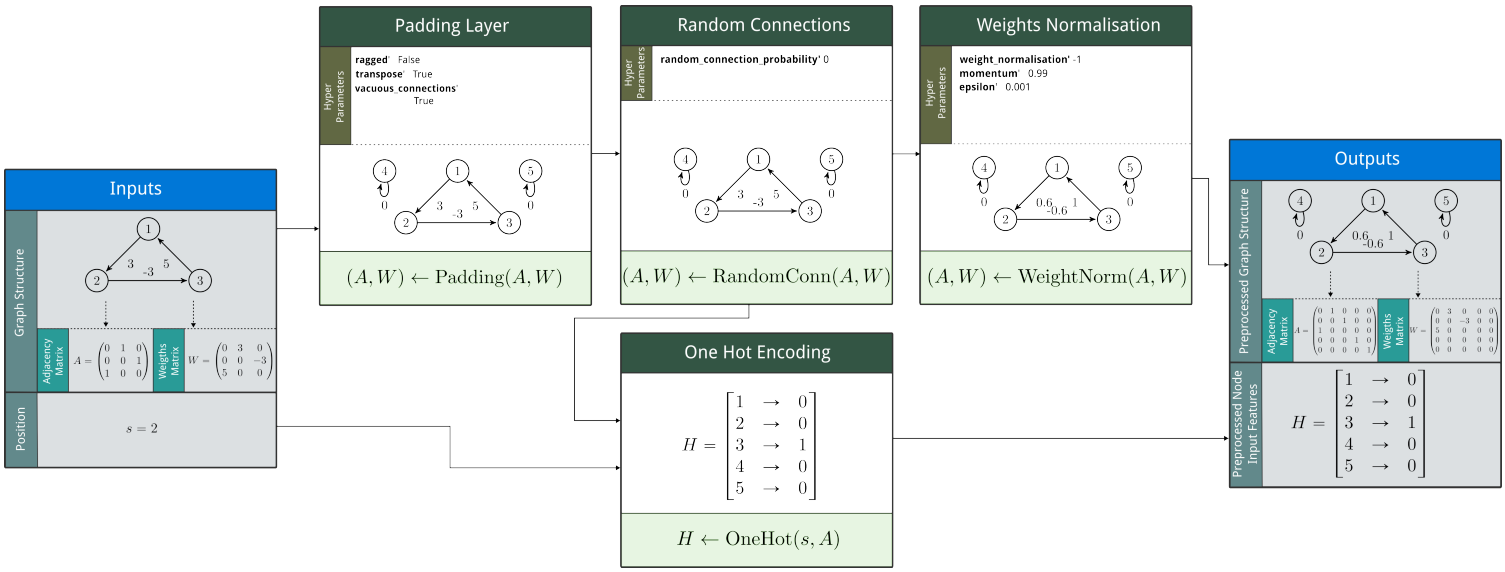


Figure 4.5: Preprocessing Block

4.5.2 Weighted Graph Convolutional Network

The weighted graph convolutional network **WGCN**, as its name suggests, is a convolutional operator acting on graphs, with many desirable properties such as "Permutation Equivariance" and "Stability under Padding".
It is based on the graph convolutional network as described on the following figure

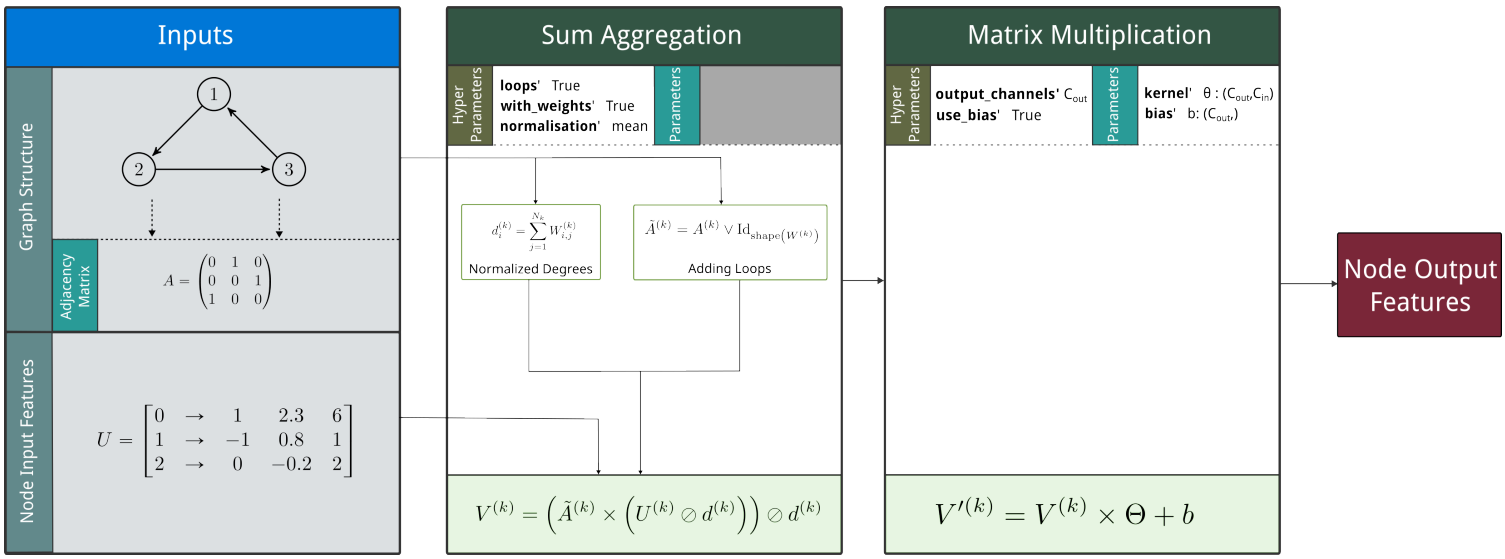


Figure 4.6: Graph Convolutional Network

Here $\oslash$ denotes the point-wise division operator.
The problem with **GCN** is that they ignore the weights information. For Mean Payoff Games, such information is crucial to determine the winner and the strategies.
For that, we introduced **WGCN** to capitalize on the weights information. The figure below shows how it is implemented.

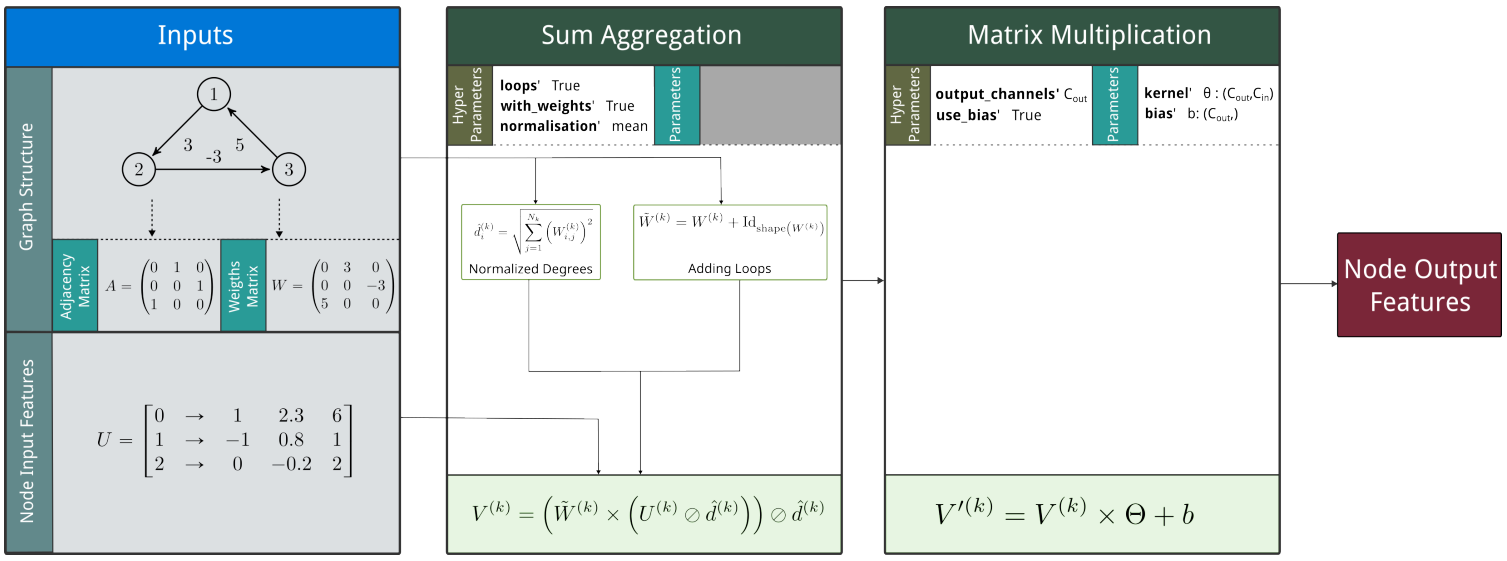


Figure 4.7: Weighted Graph Convolutional Network

The **WGCN** operator exhibits the desired properties described on 4.3, except property 4.3.3, the latter is already verified as a result of the preprocessing layer.
We were not able to find a reference to **WGCN** in the literature. The closest thing we have found that a PyTorch based GNN library named **Geometric** has an implementation for **GCN** supporting weighted graphs. Interestingly, we only differ to them by the normalization of the degrees.
As we are using *TensorFlow*, we cannot use their approach, so we had to implement **WGCN** from scratch.

4.5.3 Convolutional Block

Each intermediate block is composed of a graph convolution, and then a batch normalisation operator, as described by the figure below:

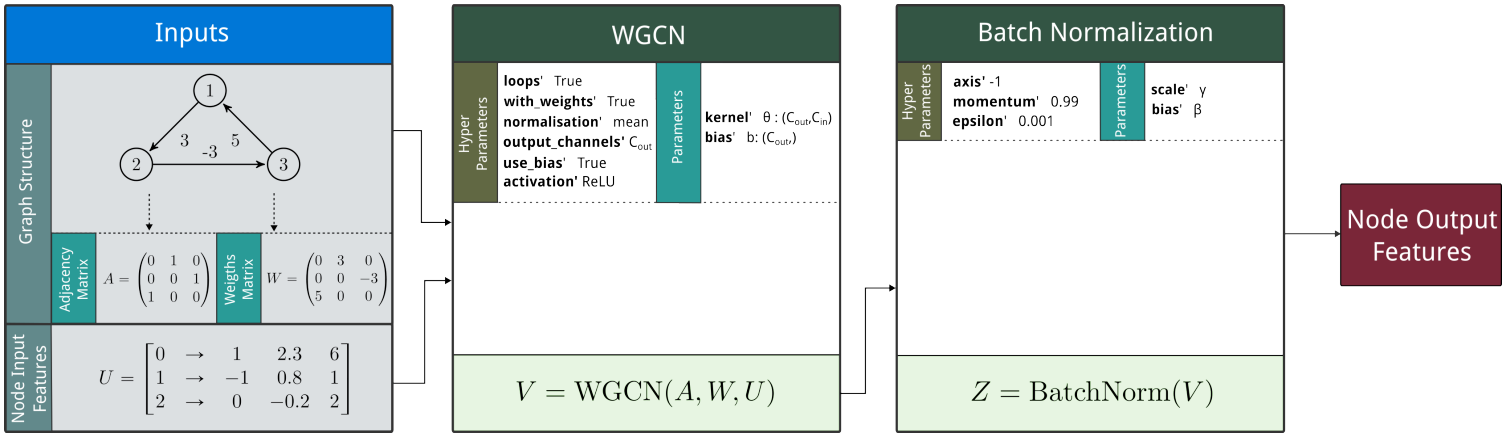


Figure 4.8: Graph Convolutional Block

This block was inspired from convolutional blocks used in computer vision models.

4.5.4 Prediction Block

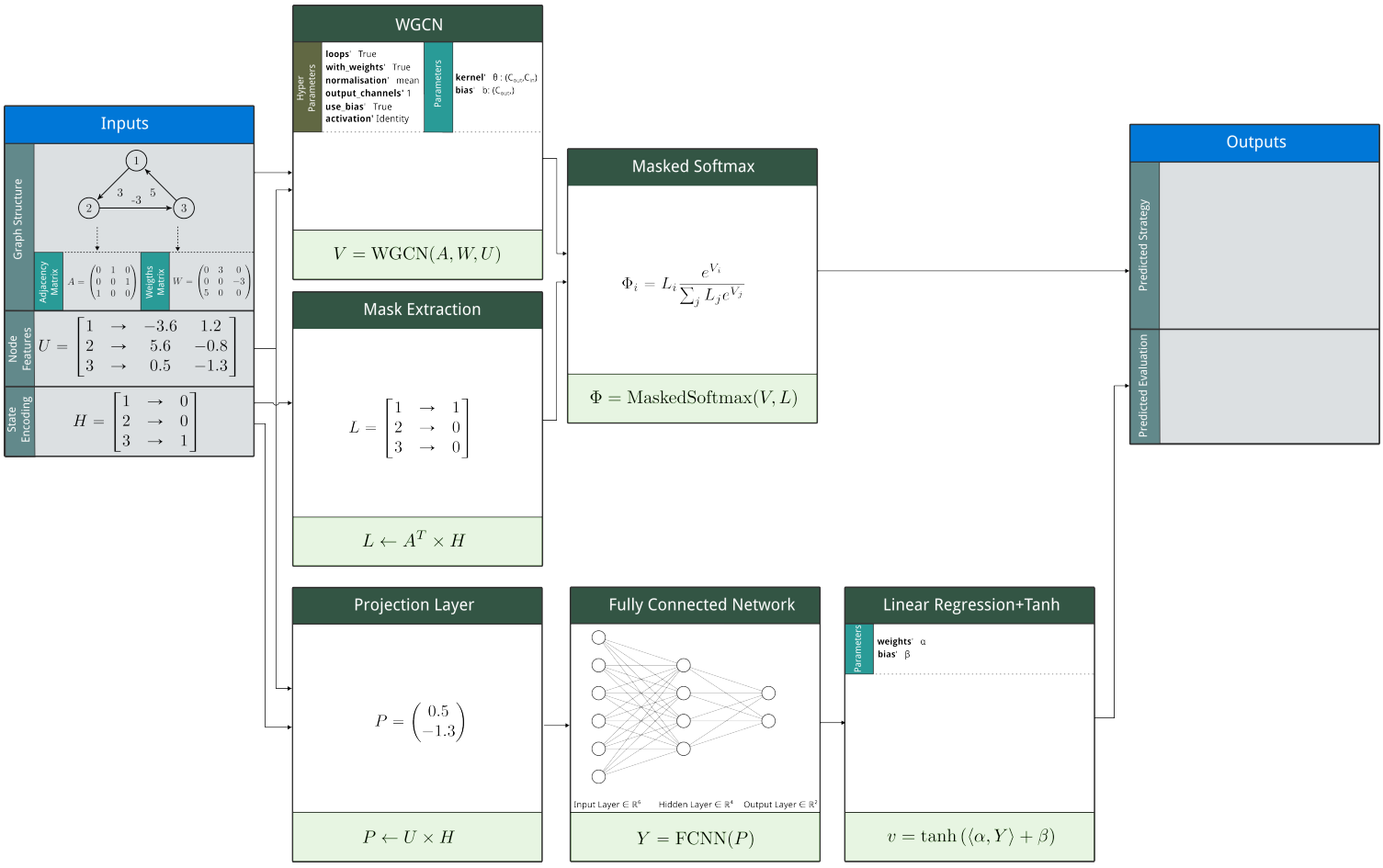


Figure 4.9: Graph Convolutional Block

4.5.5 Model Architecture

Putting all building block together, we designed a model that takes arbitrary graphs, encode them, predicts the strategy and the evaluation of the position.
We were very careful in the design so that we can verify all the desired properties described in section 4.3.
The figure below shows the whole architecture:

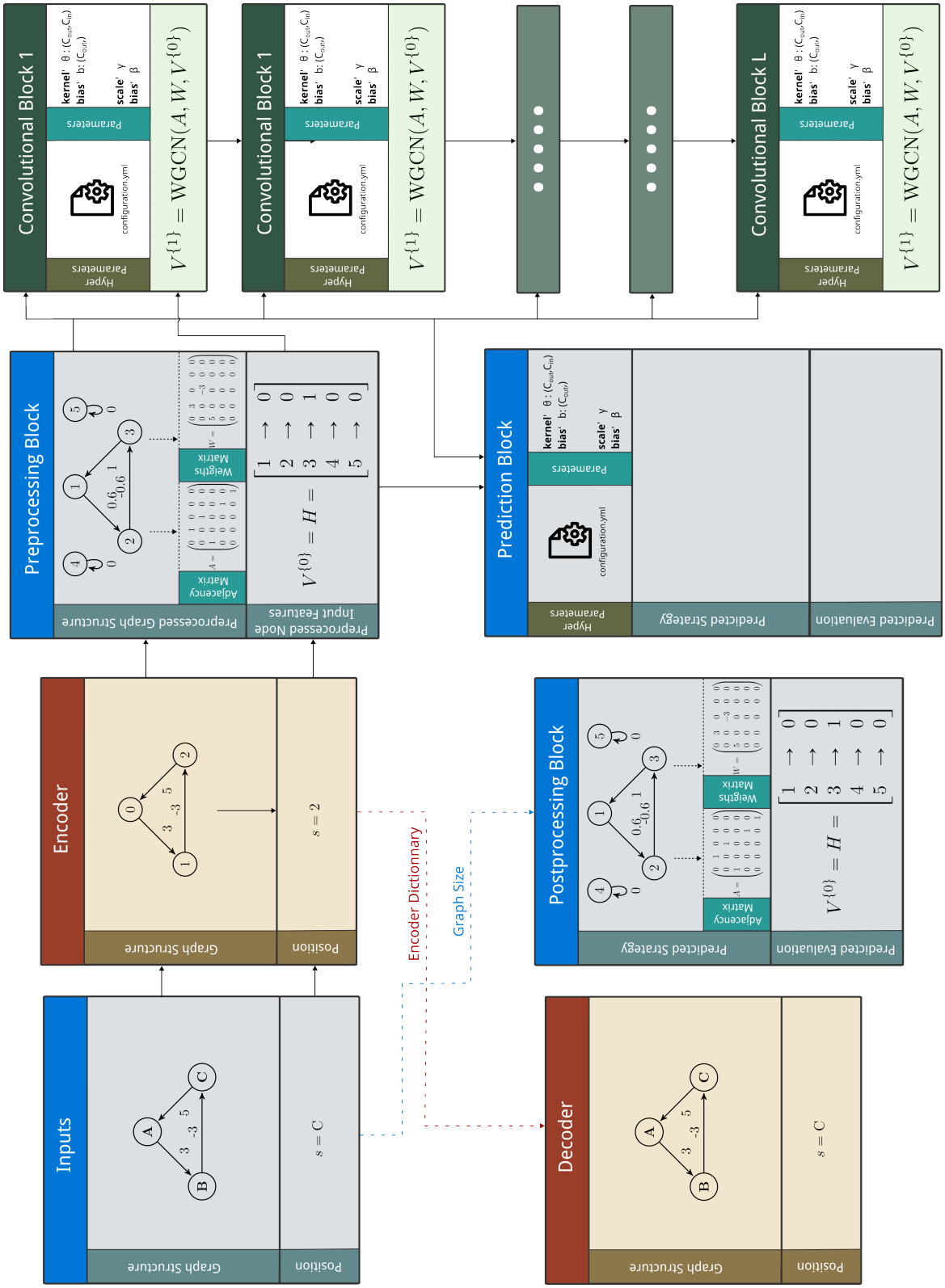


Figure 4.10: Model Architecture

Chapter 5

Reinforcement Learning Approach

5.1 Introduction

5.2 Monte Carlo Augmentation

5.2.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a crucial algorith [43]

5.3 Pipeline

5.4 Services

5.5 Configuration

Chapter 6

Analyse

For $x \in \mathcal{X}, Y \subseteq \mathcal{X}^m, c \in I$, let $\mathrm{MA}(x, Y, c)$ be defined as follow:

$$\mathrm{MA}(x, Y, c) \iff x \leqslant \max Y + c$$

New York (NY) Hi

Conclusion

Durant ce stage, nous avons étudié les BNNs, implémenté une bibliothèque de BNNs basée sur tensorflow et larq qu'on a nommé binaryflow.

Dans le premier chapitre, nous avons présenté **dB Sense** et ses activités.

Dans le deuxième chapitre, nous avons présenté le problème de la grande complexité, introduit le concept des BNNs et proposé **binaryflow** comme notre solution pour les BNNs

Dans le troisième chapiter, nous avons formalisé les BNNs, et nous avons décris leurs optimisations possible, et les problèmes rencontrés dans leur implémentation.

Dans le quatrième chapitre, nous avons présenté des modèles BNNs, chacun en dérivant ses formules

Dans le cinquième chapitre, nous avons donné notre implémentation de **binaryflow** en jusitifiant les paradigmes utilisés

Dans le sixième chapitre, nous avons analysé 3 jeux de données en implémentant des modèles BNNs pour chacune de ces 3 jeux de données, et en comparant les performances de prédicition et la complexité temps et mémoire des modèles entraînés.

Notre travail n'est qu'une petite introduction des BNNs. En effet, la liste des BNNs proposés dans la littérature est très vaste, et il existe plusieurs autres approches pour faciliter l'entraînement et l'interférence des BNNs que nous n'avons pas considéré vu les contraintes de stage, y parmi:

1. Les méthodes d'optimisations discrètes
2. Les optimiseurs dédiés au BNNs
3. Les binarisations entraînables
4. Les méthodes ensemblistes pour régulaliser les BNNs

De plus, nous avons réussi à vérifier l'optimisation de la multiplication matricielle (L'éxecution est parfois 30 fois plus rapide), mais nous n'avons pas pu intégrer cette optimisation aux modèles tensorflow. Et malgré que larq supporte lui même un déploiement optimisé, nous n'avons pas aussi pu l'exploiter puisque ce déploiement ne supporte que les processeurs ARMv8, et nous utilisons une machine avec un processeur d'architecture x86-64.

Finalement, nous avons fait une petite intégration du code carbone comme une mésure de coût d'entraînement. Une fois le problème de déploiement est résoulu, nous recommenderions l'utilisation de ce même métrique pour estimer le coût d'interférence qui va justifier l'utilisation des BNNs.

Appendix A

On Constraint Satistfaction Problems

In the previous chapters, we described how the system works, without formalising the CSP approach.
On this chapter, we will describe the CSP systems that we have used, with an equivalence proof between them.

A.1 Constraint Satisfaction Problem

A.1.1 Definition

A constraint satisfaction problem

A.1.2 Assignment

An assignment of a $\texttt{CSP}(\mathcal{X}, D, \Gamma)$ is a function $X : \mathcal{X} \to D$ such that, by replacing each $x \in \mathcal{X}$, by $X(x)$, all the constraints will evaluate to **True**

A.1.3 Polymorphism

A function $F : \mathscr{F}(\mathcal{X}, D)^k \to \mathscr{F}(\mathcal{X}, D)$ is said to be a polymorphism if:

$\forall X_1, \ldots, X_k$ assignments of $\texttt{CSP}(\mathcal{X}, D, \Gamma)$, $\quad F(X_1, \ldots, X_k)$ is also an assignment of $\texttt{CSP}(\mathcal{X}, D, \Gamma)$

Now, in the next section, we will define an important class of CSPs that is used to solve Mean Payoff Games, with the polymorphisms required for the solution algorithm's correctness

A.2 Ternary Max Atom Systems

A.2.1 Definition

- Let $\mathcal{X}$ be a finite set of variables
- Let $D = I \cup \{-\infty\}$, with $I \subseteq \mathbb{R}$.
- For $x, y, z \in \mathcal{X}, c \in I$, let $\mathrm{MA}_3(x, y, z, c)$ be defined as follow:

$$\mathrm{MA}_3(x, y, z, c) \iff x \leqslant \max(y, z) + c$$

A ternary max atom system is $\texttt{CSP}(D, \Gamma)$ where:

$$\begin{aligned}&\Gamma = \{\mathrm{MA}_3(x, y, z, c), \quad (x, y, z, c) \in \mathscr{R}\}\\&\mathscr{R} \subseteq \mathcal{X}^3 \times I\\&\mathscr{R}\text{is finite}\end{aligned}$$

A.2.2 Example

An example of a ternary max atom system is the following $\texttt{CSP}(D, \Gamma)$ with $D = \mathbb{Z}$ and Γ represented as follow:

$$\begin{aligned}x &\leqslant \max(y, z) - 1\\y &\leqslant \max(z, x) - 1\\z &\leqslant \max(x, y) - 1\end{aligned}$$

A.3 Max Atom Systems

A.3.1 Definition

- Let $\mathcal{X}$ be a finite set of variables
- Let $D = I \cup \{-\infty\}$, with $I \subseteq \mathbb{R}$
- For $x \in \mathcal{X}, Y \subseteq \mathcal{X}^m, c \in I$, let $\mathrm{MA}(x, Y, c)$ be defined as follow:

$$\mathrm{MA}(x, Y, c) \iff x \leqslant \max Y + c$$

A max atom system is $\texttt{CSP}(D, \Gamma)$ where:

$$\begin{aligned}&\Gamma = \{\mathrm{MA}(x, Y, c), \quad (x, Y, c) \in \mathscr{R}\}\\&\mathscr{R} \subseteq \mathcal{X} \times (\mathscr{P}(\mathcal{X})\backslash\{\varnothing\}) \times I\\&\mathscr{R}\text{is finite}\end{aligned}$$

A.3.2 $\mathbf{MA} \leqslant \mathbf{MA}_3$

- Let $S = \texttt{CSP}(\mathcal{X}, D, \Gamma)$ a max atom system.
- Let $R \in \Gamma$
- Let $x \in \mathcal{X}, Y \in \mathscr{P}(\mathcal{X}), c \in I$ such that $R = \mathrm{MA}(x, Y, c)$ such that $|Y| > 2$

Recursive Reduction

We will reduce the arity of R as follow:

- Let $y, z \in Y$ such that $y \neq z$
- We introduce a variable $w \notin \mathcal{X}$
- Let $\mathcal{X}' = \mathcal{X} \cup \{w\}$

- Let $Y' = (Y \cup \{w\}) \backslash \{y, z\}$
- Let $R' = \mathrm{MA}(x, Y', c)$
- Let $R_w = \mathrm{MA}(w, \{y, z\}, 0)$
- Let $\Gamma' = (\Gamma \cup \{R', R_w\}) \backslash \{R\}$
- Let $S' = \mathtt{CSP}(\mathcal{X}', D, \Gamma)$

We will prove that S' is equivalent to S.

Implication Let $X : \mathcal{X}' \to D$ an assignment of S'. It is trivial that by removing $X(w)$, $X_{|\mathcal{X}}$ is an assignment of S

Equivalence

- Let $X : \mathcal{X}' \to D$ such that $X_{|\mathcal{X}}$ is an assignment of S.
- We will set $X(w) = \max(X(y), X(z))$

Then, X is an assignment of S'

Induction

Since the number of variables is finite, the arity of each constraint is finite. Also, as the the number of constraints is finite, Applying such reduction iteratively will eventually give a system S^* equivalent to S with:

- $\mathcal{X}^*$ the set of variables with $\mathcal{X} \subseteq \mathcal{X}^*$
- Γ^* is the set of constraints:
- Each constraint is of the form $\mathrm{MA}(x, Y, c)$ with $x \in \mathcal{X}^*, Y \subseteq \mathcal{X}^*, c \in I$ with $|Y| \leqslant 2$

Now such system can be transformed to a ternary system S_3 as follow:

- The set of variables is $\mathcal{X}^*$
- The domain is D
- For every relation $R = \mathrm{MA}(x, Y, c)$ we map it to the relation $R_3 = \mathrm{MA}(x, y, z, c)$ as follow:
  - If $|Y| = 2$, then y, z are the elements of Y.
  - Otherwise, $|Y| = 1$, and $y = z$ are the same element of the singleton[1] Y.

It is trivial that S^* is equivalent to S_3. With that, S is equivalent to S_3.

[1] A set with only one element

Algorithm 10 Converting a Max Atom System to Ternary Max Atom System

Require: S an N-ary Max Atom system
Ensure: S' a ternary Max Atom system
  $S' \leftarrow \varnothing$
  $H \leftarrow \varnothing$ ▷ H is a symmetric map between variable,variable to variables
  $V \leftarrow \text{Variables}(S)$ ▷ V is a set containing all variables
  for $\mathcal{C} \in S$ **do** ▷ Iterate over constraints
    c is the constant in the right hand side of $\mathcal{C}$
    Y is the variables in the right hand side of $\mathcal{C}$
    x is the variable in the left hand side of $\mathcal{C}$
    while $|Y| > 2$ **do**
      $y \leftarrow \text{pop}(Y)$
      $z \leftarrow \text{pop}(Y)$
      if $(y, z) \notin \text{domain}\, H$ **then**
        $w \leftarrow \text{newVariable}(V)$ ▷ Generate a new formal variable not included in V
        $V \leftarrow V \cup \{w\}$
        $H(y, z) \leftarrow w$
        $H(z, y) \leftarrow w$
      end if
      $w \leftarrow H(y, z)$
      $S' \leftarrow S' \cup \{\text{MA}(w, y, z, c)\}$
      $Y \leftarrow Y \cup \{w\}$
    end while
  end for
  return S'

A.3.3 Polymorphisms

Two main family of polymorphisms are defined for Max Atom systems:

- The max polymorphisms M^k defined by:

$$M^k(X_1, \ldots, X_k)(x) = \max_{k \in \{1,\ldots,k\}} X_k(x)$$

- The translation polymorphisms T_c defined by:

$$T_c(X)(x) = X(x) + c$$

A.4 Min-Max System

- Let $\mathcal{X}$ be a finite set of variables
- Let I be the domain of the variables.
- Let $D = I \cup \{-\infty\}$, with $I \subseteq \mathbb{R}$
- For $x \in \mathcal{X}, Y \subseteq \mathcal{X}^m, C \in I^m$, let $\mathrm{MA}(x, Y, C)$ be defined as follow:

$$\mathrm{MA}(x, Y, c) \iff x \leqslant \max(Y + C)$$

- For $x \in \mathcal{X}, Y \subseteq \mathcal{X}^m, C \in I^m$, let $\mathrm{MI}(x, Y, C)$ be defined as follow:

$$\mathrm{MI}(x, Y, C) \iff x \leqslant \min(Y + C)$$

A min-max system is $\texttt{CSP}(D, \Gamma)$ where:

$$\begin{aligned} \Gamma &= \{O(x, Y, C), \quad (O, x, Y, C) \in \mathscr{R}\} \\ \mathscr{R} &\subseteq \{\mathrm{MA}, \mathrm{MI}\} \times \mathcal{X} \times (\mathcal{X} \times I)^+ \\ \mathscr{R} &\text{ is finite} \end{aligned}$$

A.4.1 Transforming to Max Atom Systems

A Max Atom system is trivially a Min Max system. So we will only prove the latter implication.
Let $S' = \texttt{CSP}(D, \Gamma)$ be a Min Max system, and let:

- Γ_{MI} be the constraints that has MI
- Γ_{MA} be the constraints that has MA

Transforming MI constraints

For each $\mathrm{MI}(x, Y, c) \in \Gamma_{\mathrm{MI}}$. we replace it with the following constraints:

$$\Gamma_{\mathrm{MI}}^{x,Y,C} = \{\mathrm{MA}(x, \{y\}, c), \quad y, c \in \mathrm{zip}(Y, C)\}$$

Tranforming MA constraints

For each $(y, c) \in \mathcal{X} \times I$ present in a max constraint of the system:

- We add a formal variable $z^{y,c}$ if $c \neq 0$.
- Else, we will simply represent by $z^{y,c}$ the variable y.

By denoting $Z^{Y,C}$ the following set:

$$Z^{Y,C} = \{z^{y,c}, \quad (y, c) \in \text{zip}(Y, C)\}$$

We build the following constraints:

$$\Gamma_{\text{MA}}^{x,Y,C} = \left\{\text{MA}(x, Z^{Y,C}, 0)\right\} \cup \{\text{MA}(z^{y,c}, \{y\}, c), \quad (y, c) \in \text{zip}(Y, C)\}$$

Building the Max Atom System

Now, let:

$$\Gamma' = \bigcup_{\text{OP} \in \{\text{MI},\text{MA}\}} \bigcup_{\text{OP}(x,Y,C) \in \Gamma_{\text{OP}}} \Gamma_{\text{OP}}^{x,Y,C}$$

The system $\texttt{CSP}(D, \Gamma')$ is an equivalent max system.

Equivalence

Let:

- $\mathcal{X}' = \mathcal{X} \cup \mathcal{X}_{\text{Generated}}$ the augmented set of variables.
- $\mathcal{C} = \text{OP}(x, Y, C) \in \texttt{CSP}(D, \Gamma)$ be a constraint.
- $X : \mathcal{X}' \to D$ an assignment of $\texttt{CSP}(D, \Gamma')$.

If OP = MI, it is trivial that $\text{OP}(x, Y, C)$ is equivalent to $\Gamma_{\text{OP}}^{x,Y,C}$.
Otherwise, for each $(y, c) \in \text{zip}(Y, C)$ we have:

$$X(z^{y,c}) \leqslant \max\{X(y)\} + c = X(y) + c$$

Now, we also have:

$$X(x) \leqslant \max_{(y,c) \in \text{zip}(Y,C)} X(z^{y,c}) + 0 \leqslant \max_{(y,c) \in \text{zip}(Y,C)} (X(y) + c)$$

With that, $X_{|\mathcal{X}}$ is an assignment of $\texttt{CSP}(D, \Gamma)$

Algorithm 11 Converting a Min-Max System to Max Atom

Require: S a Min-Max system
Ensure: S' an N-ary Max Atom system
  $S' \leftarrow \varnothing$
  $H \leftarrow \varnothing$ ▷ H is a map between variable,offsets to variables
  $V \leftarrow \text{Variables}(S)$ ▷ V is a set containing all variables
  for $\mathcal{C} \in S$ **do** ▷ Iterate over constraints
    C is the constants in the right hand side of $\mathcal{C}$
    Y is the variables in the right hand side of $\mathcal{C}$
    x is the variable in the left hand side of $\mathcal{C}$
    if $\mathcal{C}$ is a min constraint **then**
      $S' \leftarrow S' \cup \{\text{MA}(x, \{y, y\}, c), \quad (y, c) \in \text{zip}(Y, C)\}$
    else
      $Z \leftarrow \varnothing$
      for $(y, c) \in \text{zip}(Y, C)$ **do**
        if $(y, c) \notin \text{domain}\, H$ **then**
          $z \leftarrow \text{newVariable}(V)$ ▷ Generate a new formal variable not included in V
          $V \leftarrow V \cup \{z\}$
          $H(y, c) \leftarrow z$
        end if
        $S' \leftarrow S' \cup \{\text{MA}(H(y, c), \{y, y\}, c)\}$
        $Z \leftarrow Z \cup \{H(y, c)\}$
      end for
      $S' \leftarrow S' \cup \{\text{MA}(x, Z, 0)\}$
    end if
  end for

Rami ZOUARI
INSAT, Tunisia

A.5 Solving Mean Payoff

A.5.1 Reduction to Min Max System

In this section, we will solve the mean payoff by converting it to a equivalent min max system. The method we use is formalised in [3], and the equivalence is also proven in [3].

Algorithm 12 Converting a Mean Payoff Game to a Min Max system

Require: G a Mean Payoff Game
Ensure: S an Min-Max system
  $E \leftarrow E(G)$ ▷ The edges of G
  $V \leftarrow V(G)$ ▷ The variables of G
  $W \leftarrow W(G)$ ▷ The weight function of G
  $P \leftarrow P(G)$ ▷ The set of player of G
  for $(u, p) \in V \times P$ **do**
    $x \leftarrow (u, p)$
    $A \leftarrow \mathrm{Adj}(x)$
    $Y = \{(a, \bar{p}), \quad a \in A\}$
    $C \leftarrow W(A)$ ▷ Calculating the weights element wise.
    if p is Max **then**:
      OP $\leftarrow$ MA
    else
      OP $\leftarrow$ MI
    end if
    $S \leftarrow S \cup \{\mathrm{OP}(x, Y, C)\}$
  end for

A.5.2 Arc Consistency

First Implementation

At first, we took the original implementation of AC3 [32, page. 171] and modify it to support the max atom system:

Algorithm 13 AC3 for Ternary Max Atom systems

Require: $\mathcal{C}$ a ternary Max Atom constraint
Require: $\nu : V \to \mathscr{P}(D)$ the admissible values function
Require: Q a Queue of pending updates
Ensure: ν update the admissible values function.

```
x ← the left-hand side of C
(y, z) ← the right-hand side variables of C
c ← the right-hand side constant C
Z ← [x, y, z]
for o ∈ {0, 1, 2} do                                   ▷ Iterate over all rotations
    admissible ← False
    x ← Z[o]
    y ← Z[(o + 1) mod 3]
    z ← Z[(o + 2) mod 3]
    μ ← ∅                                              ▷ Set of values to delete
    for a ∈ ν(x) do
        for (b, c) ∈ ν(y) × ν(z) do
            T ← [a, b, c]
            p ← T[(−o) mod 3]
            q ← T[(1 − o) mod 3]
            q ← T[(2 − o) mod 3]
            if p ⩽ max(q, r) + c then
                admissible ← True
            end if
        end for
        if not admissible then
            μ ← μ ∪ {a}
        end if
    end for
    if |μ| > 0 then
        ν(x) ← ν(x)\μ
        append(Q, Adj x)
    end if
end for
```


Rami ZOUARI
INSAT, Tunisia

This version was very slow, took a considerable time even for Mean Payoff Games with less than 20 vertices.

Refinement

We were able to make many simplifications to the algorithm by taking advantage of the symmetry[2] of the system and the fact that it has translations and maximum as polymorphisms [5]:

Algorithm 14 AC3 Optimized for Ternary Max Atom systems

Require: $\mathcal{C}$ a ternary Max Atom constraint
Require: $\nu : V \to D$ the maximum admissible value for each variable.
Require: Q a Queue of pending updates
Require: $L = \inf D$ the smallest admissible value
Ensure: ν update the maximum admissible values function.
  $x \leftarrow$ the left-hand side of $\mathcal{C}$
  $(y, z) \leftarrow$ the right-hand side variables of $\mathcal{C}$
  $c \leftarrow$ the right-hand
  $r \leftarrow \max(\nu(y), \nu(z)) + c$
  if $r < L$ **then**
    $r \leftarrow -\infty$
  end if
  if $r < \nu(x)$ **then**
    $\nu(x) \leftarrow r$
    append(Q, x)
  end if

[2]max is a symmetric function: $\max(x, y) = \max(y, x)$

Rami ZOUARI
INSAT, Tunisia

Appendix B

On Random Mean Payoff Graphs

In the previous chapters, we gave a rough analysis of graph generation.
In this chapter, we will dive into a more detailed analysis.

B.1 Introduction

B.2 Sinkless $\mathcal{D}(n,p)$ Graph

B.2.1 Property

Let $\mathtt{P}$ be the property[1] "Graph has not sink".
This property is increasing in the sense that:

$$\forall H \text{ spanning subgraph of } G, \quad H \in \mathtt{P} \implies G \in \mathtt{P}$$

As a consequence:

$$\forall n \in \mathbb{N}, p, p' \in [0,1]/ \quad p \leqslant p', \quad \mathscr{P}(\mathcal{D}(n,p) \in \mathtt{P}) \leqslant \mathscr{P}(\mathcal{D}(n,p') \in \mathtt{P})$$

We will be interested in two properties:

- The property "Vertex v has no sinks". We denote it by $\text{NoSink}(v)$.
- The property "Graph G has no sinks at all". We denote it by $\text{Sinkless}(G)$.

B.2.2 Basic Comparison with Normal $\mathcal{D}(n,p)$

We will calculate the expected value of $\deg v$. By applying the law of total expectancy:

$$\begin{aligned}\mathbb{E}[\deg v] &= \mathbb{E}[\deg v \mid \deg v > 0] \times \mathscr{P}(\deg v > 0) + \mathbb{E}[\deg v \mid \deg v = 0] \times \mathscr{P}(\deg v = 0) \\ &= \mathbb{E}[\deg v \mid \text{Sinkless}(G)] \times \mathscr{P}(\text{NoSink}(v))\end{aligned}$$

[1] Formally, a property is a just a set of graphs. In practice, it is a set that has desirable "properties".

With that:

$$\mathbb{E}[\deg v \mid \text{Sinkless}(G)] = \frac{\mathbb{E}[\deg v]}{\mathscr{P}(\text{NoSink}(v))} = \frac{np}{1-(1-p)^n} \leqslant \frac{np}{1-e^{-1}}$$
$$\mathbb{E}[|\mathcal{E}|] = \sum_{v \in V} \mathbb{E}[\deg v \mid \text{Sinkless}(G)] = \frac{n^2 p}{1-(1-p)^n} \leqslant \frac{n^2 p}{1-e^{-1}}$$

This shows that the conditional distribution does inflict a small multiplicative bias on the expected number of edges and expected degree.
This serves as an evidence that $\mathcal{D}^S(n,p)$ is similar enough to $\mathcal{D}(n,p)$

B.2.3 Property Probability

- Let $G \sim \mathcal{D}(n,p)$
- Let v a vertex of G

The probability that $\text{NoSink}(v)$ occurs is:

$$\begin{aligned}\mathcal{P}(\text{NoSink}(v)) &= 1 - \mathcal{P}(\text{Adj}\, v = \varnothing) \\ &= 1 - \mathcal{P}(\deg v = 0) \\ &= 1 - (1-p)^n\end{aligned}$$

Now, it is clear that the sequence of events $(\text{NoSink}(v))_{v \in V}$ is independent.
With that, the probability that the whole graph is sinkless is:

$$\begin{aligned}\mathcal{P}(\text{Sinkless}(G)) &= \mathcal{P}(\text{Adj}\, v \neq \varnothing \quad \forall v \in V) \\ &= \mathcal{P}\left(\bigwedge_{v \in V} \text{NoSink}(v)\right) \\ &= \prod_{v \in V} \mathcal{P}(\text{NoSink}(v)) \\ &= (1-(1-p)^n)^n\end{aligned}$$

B.2.4 Asymptotic Analysis For Dense $\mathcal{D}(n,p)$

Let $c > 0$. We have for large enough n:

$$(1-p)^n \leqslant \frac{c}{n}$$

Which implies:

$$(1-\tfrac{c}{n})^n \leqslant (1-(1-p)^n)^n \leqslant 1$$

If we take the limit, we have:

$$e^{-c} \leqslant \lim_{n \to +\infty} (1-(1-p)^n)^n \leqslant 1 \quad \forall c > 0$$

By tending c to 0, we get:

$$\lim_{n \to +\infty} (1-(1-p)^n)^n = 1$$

B.2.5 Asymptotic Analysis For Sparse $\mathcal{D}(n,p)$

Let:

$$\begin{aligned} f : \mathbb{R}_+^* \times \mathbb{R}_+ \times \mathbb{R} &\to \mathbb{R}_+ \\ x, k, c &\to (1 - g(x,k,c))^x \\ g : \mathbb{R}_+^* \times \mathbb{R}_+ \times \mathbb{R} &\to \mathbb{R}_+ \\ x, k, c &\to \left(1 - \frac{k \ln x + c}{x}\right)^x \end{aligned}$$

By construction, $f(n,k,c)$ is the probability of a graph following $\mathcal{G}(n, \frac{k \ln n + c}{n})$ to contain no sink.

We have:

$$\begin{aligned} \ln g(k,x,c) &= x \ln \left(1 - \frac{k \ln x + c}{x}\right) \\ &= -k \ln x - c - \frac{(k(\ln x) + c)^2}{2x} + o\left(\frac{(\ln x)^3}{x^2}\right) \\ \implies g(x,k,c) &= \exp\left(-k \ln x - c - \frac{(k \ln x + c)^2}{2x} + o\left(\frac{(\ln x)^3}{x^2}\right)\right) \\ &= \frac{e^{-c}}{x^k} \times e^{\frac{-(k \ln x + c)^2}{2x} + o\left(\frac{(\ln x)^3}{x^2}\right)} \\ &= \frac{e^{-c}}{x^k}\left(1 - \frac{(k \ln x + c)^2}{2x} + o\left(\frac{(\ln x)^3}{x^2}\right)\right) \\ &= \frac{e^{-c}}{x^k} - e^{-c}\frac{k^2(\ln x)^2}{2x^{k+1}} + o\left(\frac{(\ln x)^3}{x^{k+2}}\right) \\ &= \frac{e^{-c}}{x^k} + o\left(\frac{1}{x^k}\right) \\ \implies 1 - g(x,k,c) &= 1 - \frac{e^{-c}}{x^k} + o\left(\frac{1}{x^k}\right) \\ \implies x \ln(1 - g(x,k,c)) &= -\frac{e^{-c}}{x^{k-1}} + o\left(\frac{1}{x^{k-1}}\right) \\ &\sim -\frac{e^{-c}}{x^{k-1}} \\ \implies f(x,k,x) &= e^{-\frac{e^{-c}}{x^{k-1}} + o(\frac{1}{x^{k-1}})} \end{aligned}$$

Now with that:

$$\lim_{x \to +\infty} x \ln(1 - g(x,k,c)) = \begin{cases} -\infty & \text{if} k \in [0,1[\\ -e^{-c} & \text{if} k = 1 \\ 0 & \text{otherwise if} k \in]1, +\infty[\end{cases}$$

Finally, we can conclude that:

$$\lim_{x\to+\infty} f(x,k) \begin{cases} 0 & \text{if} k \in [0,1[\\ e^{-e^{-c}} & \text{if} k = 1 \\ 1 & \text{otherwise if} k \in]1,+\infty[\end{cases}$$

B.3 Repeating Construction

B.3.1 Estimating Complexity

Now the method of rejecting graphs that have sinks and retrying give us a natural question about how many times will the algorithm reject graph until finding a desirable one.
The number of such rejections will follow a geometric law $\mathcal{G}(h(n,p))$ where:

$$h(n,p) = \mathscr{P}\left(\text{Sinkless}(\mathcal{D}(n,p))\right) = (1-(1-p)^n)^n$$

With that, the expected complexity of the algorithm will be:

$$\tilde{\mathcal{O}}\left(\frac{C(n,p)}{h(n,p)}\right) = \tilde{\mathcal{O}}\left(\frac{C(n,p)}{(1-(1-p)^n)^n}\right)$$

With $C(n,p)$ the cost of building the graph, depending on the algorithm[2].

B.3.2 Dense Graph case

Now it is clear for dense enough graphs, in particular with $p(n) \geqslant \frac{k\ln(n)}{n}$ for large enough n, the expected complexity will reduce to $\mathcal{O}(C(n,p))$. And thus, we consider the rejection method to be efficient.

B.3.3 Sparse Graph case

If $p(n) = \frac{k\ln n}{n} + c$ with $k < 1$. We have:

$$(1-(1-p)^n)^n = e^{-e^{-c}x^{1-k}+o(x^{1-k})}$$

With that, the expected complexity of the rejection method will be:

$$\tilde{\mathcal{O}}\left(C(n,p) \times \exp\left(e^{-c}x^{1-k} + o(x^{1-k})\right)\right)$$

which is an exponential algorithm, and thus inefficient for large graphs. Since property `P` is increasing, this argument generalises to $p(n) \leqslant \frac{k\ln n}{n} + c$ for large enough n

[2]The two algorithms that we have discussed are the naive $\mathcal{O}(n^2)$ algorithm and the more optimized $\mathcal{O}(pn^2+n)$ algorithm.

B.4 Binomial Rejection Construction

Instead of throwing the whole graph at once. For every vertex $u \in V$, we try to construct the adjacenty vertices of u, and repeat if the procedure gives $\operatorname{Adj} u = \varnothing$
With this trick, the expected complexity will reduce for both algorithms to:

$$\tilde{\mathcal{O}}\left(\frac{C(n,p)}{1-(1-p)^n}\right)$$

Now for our case, it is natural to assume that $p(n) \geqslant \frac{1}{n}$, as a Mean Payoff Graph does not have a sink. With that:

$$1-(1-p)^n \geqslant 1-(1-\tfrac{1}{n})^n \geqslant 1-e^{-1}$$

Therefore, the expected complexity will simplify to:

$$\tilde{\mathcal{O}}\left(C(n,p)\right)$$

Moreover, the cost of the conditionning makes only a constant $\frac{1}{1-e^{-1}} \approx 1.582$ factor slowdown, which is effectively neligible.

B.5 Expected Mean Payoff

B.5.1 Definition

- Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a mean-payoff game
- For $u \in \mathcal{V}$, Let $\mathscr{P}(u)$ be the set of probability distributions over the set $\operatorname{Adj}(u)$
- We define a fractional strategy as a function $\Phi \in \mathscr{P}$

B.5.2 Matrix Form

- Let $n = |\mathcal{V}|$
- Let $u_1, \ldots, u_n$ an enumeration of elements of $\mathcal{V}$ A fractional strategy can be represented as a matrix A such that:

$$\mathcal{P}(\Phi(u_i) = u_j) = A_{i,j}$$

Appendix C

On Probabilistic Strategies

In the previous chapters, we gave a rough analysis of graph generation.
In this chapter, we will dive into a more detailed analysis.

C.1 Markovian Nature

C.1.1 Fixing Π^{Opt}

C.1.2 Fixing both Π^{Max} and Π^{Min}

C.2 Expected Reward of a MRP

C.2.1 Markov Reward Process

Definition

Let $\mathcal{R} = (V, E, W, A, \gamma)$ be a discrete markov reward model:

- V is a finite set of states
- $E \subseteq V \times V$ is a finite set of edges
- $W : E \to \mathbb{R}$ is the weights function
- $A : E \to [0, 1]$ is the transition function, satifying:

$$\sum_{v \in \mathrm{Adj}\, u} A(u, v) = 1 \tag{C.1}$$

- $\gamma \in [0, 1]$ is the discount factor

Such process models a markov chain where from a given state u, a transition $(u, v) \in E$ occurs with probability $A(u, v)$, and gives a reward of $W(u, v)$.
In this section, we will extend both A and W to $V \times V$ by requiring that:

$$\forall e \notin E, \quad A(e) = W(e) = 0$$

Execution and Award

The execution is formalized as follow:

- Fix $X_0 = s \in V$
- For $n \in \mathbb{N}^*$, X_n will be chosen randomly from the discrete set $\operatorname{Adj} X_{n-1}$ using probabilities from A

The cumulative (discounted) reward[1] $R_\gamma(s)$ is defined as:

$$R_\gamma(s) = \sum_{n\in\mathbb{N}} \gamma^n W(X_n, X_{n+1}) \tag{C.2}$$

This term converges for $\gamma \in [0, 1[$.

For undiscounted rewards[2], such reward may not converge.
For that the average time reward $\bar{R}(s)$ starting from s, is defined as follow:

$$\bar{R}(s) = \lim_{n\to+\infty} \frac{1}{n} \sum_{k=0}^{n-1} W(X_k, X_{k+1}) \tag{C.3}$$

C.2.2 Expected discounted reward

The discounted reward is used to calculate the rewards, where in each step, the weight of that rewards decay by γ.
Here we will assume $\gamma \in [0, 1[$.

$$\begin{aligned}
\mathbb{E}\left[R_\gamma(u)\right] &= \mathbb{E}\left[\mathbb{E}\left[R_\gamma(X_0) \mid X_1\right]\right] \\
\mathbb{E}\left[R_\gamma(u)\right] &= \sum_{v\in \operatorname{Adj}\ u} \mathcal{P}(X_1 = v \mid X_0 = u) \times \left(W(u,v) + \gamma\mathbb{E}\left[R_\gamma(v)\right]\right) \\
&= \sum_{v\in V} A(u,v) \times \left(W(u,v) + \gamma\mathbb{E}\left[R_\gamma(v)\right]\right)
\end{aligned}$$

Now, by considering S_n, A, W as matrices, the equation can be simplified to[3] :

$$\mathbb{E}\left[R_\gamma\right] = \gamma A\mathbb{E}\left[R_\gamma\right] + (A \odot W)\,\mathbf{1}$$

Here:

- $\odot$ is the point-wise matrix product, and
- $\mathbf{1} = (1, \ldots, 1)^T$ is the vector of ones.

With that, the discounted reward is calculated as follow:

$$\mathbb{E}\left[R_\gamma\right] = (I - \gamma A)^{-1}(A \odot W)\,\mathbf{1} \tag{C.4}$$

[1] "reward" and "payoff" here as synonymous. We use the word "reward"
[2] Which means $\gamma = 1$.
[3] γ has to be less then 1, as otherwise we cannot generally invert the matrix $I - \gamma A$.

C.2.3 Expected average-time reward

While the dicounted reward converges for all $\gamma \in [0, 1[$. It may fail to converge in general for $\gamma = 1$. For that, we will use the average-time reward. Such metric is more informing as it does not prioritize earlier rewards. Instead, all the rewards have the same weight and the mean calculation.
In the other hand, while the intuition behind such metric is clear, it is more challenging to analyse its convergence, and calculate it directly. And this is exactly what we will do next.

Deriving Formula

$$\begin{aligned}
\mathbb{E}\left[S_n(u)\right] &= \mathbb{E}\left[\mathbb{E}\left[S_n(X_0) \mid X_1\right]\right] \\
\mathbb{E}\left[S_n(u)\right] &= \sum_{v \in \underset{u}{\text{Adj}}} \mathcal{P}(X' = v \mid X = u) \times \left(W(u,v) + \mathbb{E}\left[S_{n-1}(v)\right]\right) \\
&= \sum_{v \in V} A(u,v) \times \left(W(u,v) + \mathbb{E}\left[S_{n-1}(v)\right]\right)
\end{aligned}$$

Now, by considering S_n, A, W as matrices, the equation can be simplified to:

$$\begin{aligned}
\mathbb{E}\left[S_n\right] &= A\mathbb{E}\left[S_{n-1}\right] + (A \odot W)\,\mathbf{1} \\
&= A\mathbb{E}\left[S_{n-1}\right] + (A \odot W)\,\mathbf{1} \\
&= \sum_{k=0}^{n-1} A^k (A \odot W)\,\mathbf{1} + A^n \mathbb{E}\left[S_0\right] \\
&= \sum_{k=0}^{n-1} A^k (A \odot W)\,\mathbf{1}
\end{aligned}$$

Now, by taking the mean, and then the limit, we have:

$$\mathbb{E}\left[\bar{R}\right] = \lim_{n \to +\infty} \frac{1}{n} \sum_{k=0}^{n-1} A^k (A \odot W)\,\mathbf{1} \tag{C.5}$$

Convergence

It is not trivial that the right-hand side of (C.5) converges.
Rothblum[31] proved that such limits exists and calculated its value. We will reformulate his results in the following theorem.

Theorem 6 *For any stochastic matrix A, the limit $\lim_{n \to +\infty} \frac{1}{n} \sum_{k=0}^{n-1} A^k$ exists, and is equal to the* ***projection*** *matrix T* ***uniquely*** *defined as follow:*

- $\operatorname{Im} T = \ker(\mathrm{Id} - A)$
- $\operatorname{Im} T^H = \ker(\mathrm{Id} - A^H)$

Now theorem 6 gives a straightforward construction of T. We only have to build a projection matrix such that:

Rami ZOUARI
INSAT, Tunisia

- It spans $\ker(\mathrm{Id} - A)$
- It cancels at $\ker T = \ker(\mathrm{Id} - A^H)^\perp = \mathrm{Im}(\mathrm{Id} - A)$

Furthermore, Rothblum[31] gave a closed-form expression that applies to such projection T:

$$T = \lim_{n \to +\infty} \frac{1}{n} \sum_{k=0}^{n-1} A^k = X(Y^H X)^{-1} Y^H \tag{C.6}$$

Where:

- X is the matrix whose column vectors span $\mathrm{Im}\, T$
- Y is the matrix whose column vectors span $\mathrm{Im}\, T^H$

C.3 Evaluation of probabilistic strategies

C.3.1 Definition

- Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a mean-payoff game
- For $u \in \mathcal{V}$, Let $\mathscr{P}(u)$ be the set of probability distributions over the set $\mathrm{Adj}(u)$
- We define a fractional strategy as a function $\Phi \in \mathscr{P}$

C.3.2 Matrix Form

- Let $n = |\mathcal{V}|$
- Let $u_1, \ldots, u_n$ an enumeration of elements of $\mathcal{V}$ A fractional strategy can be represented as a matrix A such that:
$$\mathcal{P}(\Phi(u_i) = u_j) = A_{i,j}$$

Bibliographie

[1] Dorit Aharonov and Oded Regev. "Lattice Problems in NP cap coNP". In: *Foundations of Computer Science, 1975., 16th Annual Symposium on* (May 2004). DOI: 10.1109/FOCS.2004.35.

[2] Xavier Allamigeon et al. "Combinatorial Simplex Algorithms Can Solve Mean Payoff Games". In: *SIAM Journal on Optimization* 24.4 (2014), 2096–2117. DOI: 10.1137/140953800. eprint: https://doi.org/10.1137/140953800. URL: https://doi.org/10.1137/140953800.

[3] Albert Atserias and Elitza Maneva. "Mean-Payoff Games and the Max-Atom Problem". In: (Jan. 2009).

[4] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation". In: *CoRR* abs/1308.3432 (2013). arXiv: 1308.3432. URL: http://arxiv.org/abs/1308.3432.

[5] Manuel Bodirsky and Marcello Mamino. *Tropically convex constraint satisfaction.* 2017. arXiv: 1506.04184 [cs.CC].

[6] Melissa Chan. *Go Player Finally Defeats Google's AlphaGo After 3 Losses.* 14 Mar 2016. URL: https://time.com/4257406/go-google-alphago-lee-sedol.

[7] Anne Condon. "The complexity of stochastic games". In: *Information and Computation* 96.2 (1992), 203–224. ISSN: 0890-5401. DOI: 10.1016/0890-5401(92)90048-K. URL: https://www.sciencedirect.com/science/article/pii/089054019290048K.

[8] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), 273–297.

[9] Matthieu Courbariaux and Yoshua Bengio. "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1". In: *CoRR* abs/1602.02830 (2016). arXiv: 1602.02830. URL: http://arxiv.org/abs/1602.02830.

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "BinaryConnect: Training Deep Neural Networks with binary weights during propagations". In: *CoRR* abs/1511.00363 (2015). arXiv: 1511.00363. URL: http://arxiv.org/abs/1511.00363.

[11] A. Ehrenfeucht and J. Mycielski. "Positional Strategies for Mean Payoff Games". In: *Int. J. Game Theory* 8.2 (1979), 109–113. ISSN: 0020-7276. DOI: 10.1007/BF01768705. URL: https://doi.org/10.1007/BF01768705.

[12] Elizabeth Gibney. "Self-taught AI is best yet at strategy game Go". In: *Nature* (Oct. 2017). DOI: 10.1038/nature.2017.22858.

[13] Citu Group. *What is the carbon footprint of a house?* URL: https://citu.co.uk/citu-live/what-is-the-carbon-footprint-of-a-house.

[14] Julian Gutierrez, Thomas Steeples, and Michael Wooldridge. "Mean-Payoff Games with ω-Regular Specifications". In: *Games* 13.1 (2022). ISSN: 2073-4336. DOI: 10.3390/g13010019. URL: https://www.mdpi.com/2073-4336/13/1/19.

[15] Koen Helwegen et al. "Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization". In: *CoRR* abs/1906.02107 (2019). arXiv: 1906.02107. URL: http://arxiv.org/abs/1906.02107.

[16] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), 1735–1780.

[17] IBM. *Présentation générale de CRISP-DM*. https://www.ibm.com/docs/fr/spss-modeler/saas?topic=dm-crisp-help-overview. 2017.

[18] A.G. Ivakhnenko et al. *Cybernetics and Forecasting Techniques*. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1967. ISBN: 9780444000200. URL: https://books.google.tn/books?id=rGFgAAAAMAAJ.

[19] Zohar Jackson. *Free Spoken Digits Dataset*. https://github.com/Jakobovski/free-spoken-digit-dataset. 2019.

[20] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2015).

[21] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. USA: Addison Wesley Longman Publishing Co., Inc., 1997. ISBN: 0201896834.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[23] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), 2278–2324. DOI: 10.1109/5.726791.

[24] Yann LeCun. *THE MNIST DATABASE*. http://yann.lecun.com/exdb/mnist. 1998.

[25] Xiaofan Lin, Cong Zhao, and Wei Pan. "Towards Accurate Binary Convolutional Neural Network". In: *CoRR* abs/1711.11294 (2017). arXiv: 1711.11294. URL: http://arxiv.org/abs/1711.11294.

[26] Seppo Linnainmaa. "Taylor Expansion of the Accumulated Rounding Error". In: *BIT* 16.2 (1976), 146–160. ISSN: 0006-3835. DOI: 10.1007/BF01931367. URL: https://doi.org/10.1007/BF01931367.

[27] Zechun Liu et al. "Bi-Real Net: Enhancing the Performance of 1-bit CNNs With Improved Representational Capability and Advanced Training Algorithm". In: *CoRR* abs/1808.00278 (2018). arXiv: 1808.00278. URL: http://arxiv.org/abs/1808.00278.

[28] Kim Martineau. *Shrinking deep learning's carbon footprint*. 7Aug 2021. URL: https://news.mit.edu/2020/shrinking-deep-learning-carbon-footprint-0807.

[29] *NP intersect coNP*. https://kintali.wordpress.com/2010/06/06/np-intersect-conp/. Accessed: 2023-08-28.

[30] Mohammad Rastegari et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". In: *CoRR* abs/1603.05279 (2016). arXiv: 1603.05279. URL: http://arxiv.org/abs/1603.05279.

[31] G. Rothblum. "Computation of the eigenprojection of a nonnegative matrix at its spectral radius". In: *Stochastic Systems: Modeling, Identification and Optimization, II*. Ed. by Roger J.-B. Wets. Berlin, Heidelberg: Springer Berlin Heidelberg, 1976, 188–201. ISBN: 978-3-642-00786-6. DOI: 10.1007/BFb0120751. URL: https://doi.org/10.1007/BFb0120751.

[32] Stuart J. Russell. *Artificial intelligence : a modern approach*. Includes bibliographical references (pages 1063-1093) and index. Third edition. Upper Saddle River, N.J. : Prentice Hall, [2010] ©2010, 2010. URL: https://search.library.wisc.edu/catalog/9910082172502121.

[33] Md. Sahidullah and Goutam Saha. "Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition". In: *Speech Communication* 54.4 (2012), 543–565. ISSN: 0167-6393. DOI: 10.1016/j.specom.2011.11.004. URL: https://www.sciencedirect.com/science/article/pii/S0167639311001622.

[34] Benjamin Sanchez-Lengeling et al. "A Gentle Introduction to Graph Neural Networks". In: *Distill* (2021). https://distill.pub/2021/gnn-intro. DOI: 10.23915/distill.00033.

[35] Sven Schewe. "An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games". In: *Computer Science Logic*. Ed. by Michael Kaminski and Simone Martini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, 369–384. ISBN: 978-3-540-87531-4.

[36] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (2017), 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270. URL: https://doi.org/10.1038/nature24270.

[37] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *CoRR* abs/1906.02243 (2019). arXiv: 1906.02243. URL: http://arxiv.org/abs/1906.02243.

[38] Tim Dettmers. *Deep Learning in a Nutshell: History and Training*. https://developer.nvidia.com/blog/deep-learning-nutshell-history-training/. 16 Dec. 2015.

[39] Aimee Wynsberghe. "Sustainable AI: AI for sustainability and the sustainability of AI". In: *AI and Ethics* 1 (Feb. 2021). DOI: 10.1007/s43681-021-00043-6.

[40] Chunyu Yuan and Sos S. Agaian. "A comprehensive review of Binary Neural Network". In: *CoRR* abs/2110.06804 (2021). arXiv: 2110.06804. URL: https://arxiv.org/abs/2110.06804.

[41] Chunyu Yuan and Sos S. Agaian. "A comprehensive review of Binary Neural Network". In: *CoRR* abs/2110.06804 (2021), 3–4. arXiv: 2110.06804. URL: https://arxiv.org/abs/2110.06804.

[42] Uri Zwick and Michael S. Paterson. "The complexity of mean payoff games". In: *Computing and Combinatorics*. Ed. by Ding-Zhu Du and Ming Li. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, 1–10. ISBN: 978-3-540-44733-7.

[43] Maciej Świechowski et al. "Monte Carlo Tree Search: a review of recent modifications and applications". In: *Artificial Intelligence Review* 56.3 (2022), 2497–2562. DOI: 10.1007/s10462-022-10228-y. URL: https://doi.org/10.1007%2Fs10462-022-10228-y.