



Projet de Fin d'Etudes

En vue de l'obtention du

Diplôme National d'Ingénieur en Sciences Appliquées et Technologie

Spécialité : Génie Logiciel

Titre

Titre

Présenté par

Prénom et Nom

Réalisé au sein de/du

Logo de l'entreprise d'accueil

Soutenu le 2023 devant le jury composé de :

..... Président(e) du jury
..... Examinateur (trice)
..... Responsable à l'entreprise
..... Responsable à l'INSAT

Année Universitaire : 2022 / 2023



Projet de Fin d'Etudes

En vue de l'obtention du

**Diplôme National d'Ingénieur
en Sciences Appliquées et Technologie**

Spécialité : Génie Logiciel

Titre

Titre

Présenté par

Prénom et Nom

Réalisé au sein de/du

Logo de l'entreprise d'accueil

Responsable à l'entreprise	Responsable à l'INSAT
Date :	Date :

Dedication

To my beloved parents, Your unwavering love, countless sacrifices, and boundless support have shaped me into the person I am today. I am forever grateful for the countless sacrifices you made to provide me with opportunities and a better life. Your guidance and encouragement have been my guiding lights, and I cherish every moment spent with you. Though I can never fully repay you, know that your love and sacrifices will always be etched in my heart.

To my family, Your love, encouragement, and solidarity have been my bedrock. Your support, both seen and unseen, has been a driving force behind my accomplishments. Each one of you has played an invaluable role in my life, and I am forever grateful for your presence and support.

To my dear friends, The friendships forged on this journey have been priceless treasures. Your laughter, companionship, and unwavering support have enriched my life in ways words cannot express. The memories we've created together are etched in my heart, and I am honored to dedicate this book to the unforgettable times we've shared. With heartfelt thanks, Oussama

Acknowledgment

Words cannot express my gratitude to Prof. Riadh Robbana, my academic supervisor, for his excellent mentorship not only throughout my thesis journey, but also my academic journey.

I am sincerely thankful to Prof. Manuel Bodirsky, my supervisor at TU Dresden, for providing me with the invaluable opportunity to research . His expertise, insight and kindness boosted my learning experience and expanded my horizons.

I am greatly thankful to Mr. Florian Starke, my co-supervisor at TU Dresden, for not only providing guidance, but also for being a great friend.

I would also like to express my recognition to my esteemed professors whose excellent courses and dedicated mentorship have equipped me with the knowledge and skills necessary for this achievement. Their commitment to education has been instrumental in my academic growth.

I am indebted to the INSAT community for fostering an environment of inspiration and growth. The collective spirit of students, faculty, and staff has continuously motivated me to strive for excellence.

Through this acknowledgment, I express my heartfelt gratitude to my friends who have been my pillars of strength.

Lastly, I would be remiss in not mentioning my family, my father Anouar, my mother Narjes and my two sisters Souha and Safa, Their belief in me has kept my spirits and motivation high during this process. I would also like to thank my cat Lara for all the entertainment and emotional support during the writing of this thesis.

With heartfelt thanks,

Rami.

Abstract

Mean Payoff Game (MPG) are being extensively investigated with the hopes of finding a general polynomial time solver. While this is a very desired goal, it is ambitious. We will use instead a Reinforcement Learning (RL) method using an underlying Deep Learning (DL) model based on Graph Neural Network (GNN) that tries to approximate solutions of a MPG instance. The learnable parameters of the model will be updated using a Self Play (SP) approach that is inspired from Alpha Zero.

To achieve all this, we will implement a MPG library, generate two MPG datasets, and then try solve and analyse them. We will then design our GNN model with high focus on keeping as much symmetries as possible in the model itself. Finally, we will implement a whole distributed SP system based on Alpha Zero that will be used for learning purposes, in the hope of getting a model that plays decently.

Contents

List of Figures	13
List of Tables	15
List of Algorithms	16
Acronyms	17
Introduction	18
1 Project context	19
1 Presentation of the host institute	19
1.1 Presentation of the institute of Algebra	19
1.2 Research interest	20
2 Project definition	20
2.1 General frame	20
2.2 Motivations	20
2.3 Presentation of the project	20
3 Requirement	21
3.1 Functional Requirements	21
3.2 Non-functional requirements	21
4 Introduction to Mean Payoff Games	23
4.1 Mean Payoff Games	23
4.2 State of the art	24
5 Introduction to learning approaches	25
5.1 Machine learning	25
5.2 Deep learning	25
5.2.1 Neural Networks	25
5.2.2 Definition	26
5.3 Reinforcement learning	26
5.4 Self Play	27
6 Introduction to high power computing	28
6.1 HPC	28
6.2 Prerequisites	28
6.3 Architecture	28
7 Work methodologies	29
7.1 Agile Methodology	29
7.2 Crisp-DM	29

2 Formalisation & Implementation	33
1 Formalisation	33
1.1 Di-Graph	33
1.2 Mean Payoff Game	34
1.3 Well Foundness	35
1.4 Properties	36
1.4.1 Two Player	36
1.4.2 Turn Based	36
1.5 Symmetries	36
1.5.1 Duality	36
1.6 Strategy	36
1.6.1 Deterministic Strategies	36
1.6.2 Probabilistic Strategies	37
1.6.3 Considered Strategies	37
1.6.4 Positional Optimal Strategy	37
1.6.4.1 Weak Optimality	37
1.6.4.2 Strong Optimality	37
1.6.4.3 Payoff Optimality	38
1.7 Mean Payoff	38
1.7.1 Mean Payoff	38
1.7.2 Convergence Issues	39
1.7.3 Game Value	39
2 Evaluating Strategies	39
2.1 Monte-Carlo Simulations	40
2.2 Positional Strategies	40
2.3 Probabilistic Strategies	41
3 Countering Strategies	42
3.1 Deterministic Counter Strategy	42
3.2 Probabilistic Counter Strategy	42
4 Library Implementation	42
4.1 mpg	43
4.1.1 csp	43
4.1.2 graphs	43
4.1.3 visualisation	44
4.1.4 games	44
4.1.5 ml	46
4.1.6 gnn	46
4.1.7 rl	46
4.1.8 sp	46
4.1.9 wrapper	46
4.2 mpgcpp	46
4.2.1 csp	46
4.2.2 games	46
4.2.3 mpgio	48
4.3 Environment	48
4.4 Testing	48
4.5 Structure	48

3	Dataset Generation & Annotation	49
1	Decomposition	49
2	Graph distributions	50
2.1	$\mathcal{D}(n, p)$ Family	50
2.2	$\mathcal{D}(n, m)$ Family	50
2.3	$\mathcal{D}(n, p)$ Graph Construction	50
2.3.1	Naive Method	50
2.3.2	Optimized Method	51
2.4	Choice Function	52
2.4.1	First Proposition	52
2.4.2	Complexity	53
2.4.3	Refinement	54
2.5	Complexity of Optimised $\mathcal{D}(n, p)$ Graph Construction	54
2.6	$\mathcal{D}(n, m)$ Construction	55
3	Sinkless Conditionning	56
3.1	Graph rejection method	56
3.1.1	Definition	56
3.1.2	Estimating Complexity	56
3.1.2.1	Dense Graph case	56
3.1.2.2	Sparse Graph case	56
3.2	Degree rejection method	57
4	Proposed MPG distributions	59
4.1	Desired properties of MPG distributions	59
4.1.1	Fairness	59
4.1.2	Symmetric	59
4.2	Implemented Distributions	59
5	MPG Generation	60
5.1	Distribution	60
5.1.1	Dense Graphs	60
5.1.2	Sparse Graphs	61
5.2	Implementation	61
5.3	Deployment	61
6	Annotation	62
6.1	Approach	62
6.2	Target Values	62
6.3	Heuristics	62
6.3.1	Linear Bound	62
6.3.2	Early Stopping	63
6.4	Implementation	64
6.4.1	Algorithm	64
6.4.2	Complexity Analysis	66
7	Deployment	66
7.1	Pipeline	66
7.2	Results	69
4	Model Design	70
1	Properties	70
1.1	Totality	71

1.1.1	Definition	71
1.1.2	Importance	71
1.2	Node Agnostic	71
1.2.1	Formalisation	71
1.2.2	Explanation	71
1.2.3	Importance	71
1.3	Invariance under Positive Scaling	72
1.3.1	Standard Scaling	73
1.3.2	Maximum Scaling	73
1.3.2.1	Implementation Notes:	73
1.4	Permutation Equivariance	73
1.4.1	Definition	73
1.4.2	Importance	73
1.5	Stability under Padding	74
1.5.1	Definition	74
1.5.2	Importance	74
1.5.3	Removing Unreachable Nodes	74
2	Considered Models	75
2.1	Value based Model	75
2.2	Strategy based Model	76
3	Building the Model	76
3.1	Preprocessing Block	76
3.1.1	Padding Layer	77
3.1.2	Random Connections	77
3.1.3	Weights Normalisation	77
3.1.4	Weights Noise	77
3.1.5	One Hot Encoding	78
3.2	Weighted Graph Convolutional Network	79
3.3	Convolutional Block	80
3.4	Prediction Block	81
3.5	Model Architecture	81
4	Optimization	83
4.1	Loss function	83
4.2	Optimizer	83
4.3	Hyper-parameters	84
5	Configuration	84
5.1	Model configuration	84
5.2	Training configuration	85
5	Reinforcement learning and Self play approach	87
1	Reinforcement Learning	87
1.1	Definition	87
1.2	Theory	88
1.2.1	Single Agent	88
1.2.2	Multi Agent	88
1.3	RL formalisation of a MPG	89
1.3.1	Instance based formalism	89
1.3.2	Global formalism	89

1.4	Benefits of global formalism	90
1.5	Generating random MPG	90
1.5.1	$\mathcal{D}_{p,US}^*$ distribution	90
1.5.2	$\mathcal{D}_{c,US}^*$ distribution	91
1.5.3	Importance	91
1.6	Considered RL algorithm	91
2	Monte Carlo Tree Search	93
2.1	Selection	93
2.2	Expansion	93
2.3	Simulation	93
2.4	Backpropagation	95
2.5	Wrapping up	95
3	Model based MCTS	96
3.1	Create children	96
3.2	Playout Policy	97
3.3	Selection Policy	98
3.4	Wrapping up	98
3.5	Updating model	98
3.5.1	Evaluation estimate	99
3.5.2	Strategy estimate	99
3.5.3	Refitting	99
4	Services	100
4.1	Rationale	100
4.2	Learner	100
4.3	Actor	101
4.4	Evaluator	101
5	Self-play pipeline	101
5.1	Decoupling service communication	101
5.2	Distributing the pipeline	102
5.3	REST servers	103
5.4	gRPC server	104
5.5	Tensorboard server	104
6	Implementation	104
7	Deployment	104
8	Configuration	105
8.1	Replay buffer	106
8.2	Services	106
9	Execution	107
6	Analyse	108
Conclusion & Perspectives		109
A	On Constraint Satisfaction Problems	111
1	Constraint Satisfaction Problem	111
1.1	Definition	111
1.2	Assignment	111
1.3	Polymorphism	111
2	Ternary Max Atom Systems	112

2.1	Definition	112
2.2	Example	112
3	Max Atom Systems	112
3.1	Definition	112
3.2	$MA \leq MA_3$	113
3.2.1	Recursive Reduction	113
3.2.1.1	Implication	113
3.2.1.2	Equivalence	113
3.2.2	Induction	113
3.3	Polymorphisms	114
4	Min-Max System	115
4.1	Transforming to Max Atom Systems	115
4.1.1	Transforming MI constraints	115
4.1.2	Transforming MA constraints	115
4.1.3	Building the Max Atom System	116
4.1.4	Equivalence	116
5	Solving Mean Payoff	118
5.1	Reduction to Min Max System	118
5.2	Arc Consistency	118
5.2.1	First Implementation	118
5.2.2	Refinement	120
B	On Random Mean Payoff Graphs	121
1	Sinkless $\mathcal{D}(n, p)$ Graph	121
1.1	Property	121
1.2	Basic Comparison with Normal $\mathcal{D}(n, p)$	122
1.3	Property Probability	122
1.4	Asymptotic Analysis For Dense $\mathcal{D}(n, p)$	122
1.5	Asymptotic Analysis For Sparse $\mathcal{D}(n, p)$	123
2	Verification of distribution properties	124
2.1	Symmetric	124
2.2	Fairness	125
2.3	Conclusion	125
3	Expected Mean Payoff	125
3.1	Definition	125
3.2	Matrix Form	125
C	On Probabilistic Strategies	126
1	Markovian Nature	126
1.1	Fixing Π^{Opt}	126
1.1.1	Transitions	126
1.1.2	Rewards	127
1.2	Fixing both Π^{Max} and Π^{Min}	127
2	Expected Reward of a MRP	127
2.1	Markov Reward Process	127
2.1.1	Definition	127
2.1.2	Execution and Award	128
2.2	Expected discounted reward	128
2.3	Expected average-time reward	129

2.3.1	Deriving Formula	129
2.3.2	Convergence	129
3	Evaluation of probabilistic strategies	130
3.1	Definition	130
3.2	Matrix Form	130
	Bibliography	134

List of Figures

1.1	A MPG instance	23
1.2	An illustration of a neural network	25
1.3	Relation between AI, ML and DL	26
1.4	An example of deep learning model	26
1.5	An example of a reinforcement learning system	27
1.6	Alpha Go pipeline	27
1.7	HPC architecture	28
1.8	Agile development cycle	29
1.9	Agile development cycle	30
1.10	Gantt diagram	31
2.1	An example of a MPG	35
2.2	An example of an execution with non-convergent Mean Payoffs	36
2.3	mpg library	43
2.4	Generated visualisation for a Mean Payoff Game with the optimal strategies	44
2.5	games module	45
2.6	mpgcpp library	46
2.7	games module	47
3.1	Experimental performance to generate dense & sparse graphs	55
3.2	Exponential blowup when using the graph rejection method	57
3.3	Small slowdown due to the degree rejection method	58
3.4	HPC pipeline to generate mean payoff graphs	61
3.5	A counter example to the Linear Bound heuristic	63
3.6	HPC pipeline to annotate mean payoff graphs	66
3.7	Activity diagram describing the behaviour of a node	68
4.1	Three isomorphic Mean Payoff Games	72
4.2	Mean Payoff Game, with a rescaled version using Maximum normalization	73
4.3	Mean Payoff Game before and after removing unreachable nodes	75
4.4	An MPG where the first player always wins.	76
4.5	Preprocessing Block	78
4.6	Graph Convolutional Network	79
4.7	Weighted Graph Convolutional Network	79
4.8	Graph Convolutional Block	80
4.9	Graph Convolutional Block	81
4.10	Model Architecture	82
4.11	Model section of the configuration file	85
4.12	Training section of the configuration file	85

5.1	A Reinforcement Learning system	88
5.2	One iteration of a MCTS algorithm	96
5.3	Main steps of Alpha Zero	100
5.4	Relation between different services	101
5.5	Class diagram of the services	102
5.6	Illustration of how Reverb works	104
5.7	Pipeline Architecture	105
5.8	Replay buffer section of the configuration file	106
5.9	Replay buffer section of the configuration file	107

List of Tables

1.1	Non functional requirements	22
2.1	Winning conditions for Mean Payoff Games	34
3.1	Le tableau d'avancement des BNNs	60
3.2	List of heuristics used by the solver	69
4.1	Le tableau d'avancement des BNNs	84
5.1	Supported routes	103

List of Algorithms

1	Deterministic strategies evaluation	41
2	Probabilistic strategies evaluation	42
3	$\mathcal{D}(n, p)$ Graph Generation	51
4	$\mathcal{D}(n, p)$ Graph Generation Optimisation	51
5	$\mathcal{D}(n, p)$ Choice without replacement	52
6	Fine tuned $\mathcal{D}(n, p)$ Choice without replacement	54
7	Fine tuned $\mathcal{D}(n, p)$ Choice without replacement	55
8	$\mathcal{D}^\bullet(n, p)$ Fast graph generation	57
9	Solving a Mean Payoff Graph for all states	65
10	Selection algorithm for MCTS	93
11	Expansion algorithm for MCTS	94
12	Simulation algorithm for MCTS	94
13	Backpropagation algorithm for MCTS	95
14	MCTS Algorithm	95
15	Create children	97
16	Alpha Zero MCTS Simulation	97
17	model-based MCTS playout policy	98
18	Converting a Max Atom System to Ternary Max Atom System	114
19	Converting a Min-Max System to Max Atom	117
20	Converting a Mean Payoff Game to a Min Max system	118
21	AC3 for Ternary Max Atom systems	119
22	AC3 Optimized for Ternary Max Atom systems	120

Acronyms

AI Artificial Intelligence. 6, 11, 15, 16

CSP Constraint Satisfaction Problem. 47

DL Deep Learning. 6, 11, 12, 15, 16

GNN Graph Neural Network. 12

gRPC gRPC Remote Procedure Calls. 4, 82, 83, 85

HPC High Power Computing. 12

Http Hyper-Text Transfer Protocol. 82

MAL Multi-Agent Learning. 17

MCTS Monte Carlo Tree Search. 3, 7, 9, 73–81

MDP Markov Decision Process. 73

ML Machine Learning Learning. 6, 11, 12, 15, 16, 72

MPG Mean Payoff Game. 2, 3, 11, 12, 14, 18, 37, 45, 69, 73, 77–80

NN Neural Network. 15

NY New York. 88

REST Representational State Tranfer. 4, 82, 83, 85

RL Reinforcement Learning. 3, 12, 16, 17, 72, 73

SMPG Stochastic Mean Payoff Game. 12

SP Self Play. 12, 17

YAML YAML Ain't Markup Language. 70, 85

Introduction

With the rising power of Artificial Intelligence (AI), and especially Deep Learning (DL), AI is being applied with a huge success to problems that were computationally intractable.

In this report, we will focus on a family of games known as Mean Payoff Game (MPG). It has been studied and analysed extensively with the hopes of finding an efficient general method to solve instances of that game. To the day of writing this report, no breakthroughs were being found.

Instead, we will try to approximate solutions using a completely different method based on Reinforcement Learning (RL), with an underlying DL algorithm using Self Play (SP) to refine its understanding of the game. Our methods will not only learn individual instances of the game, but also generalise to a whole class of MPG

In the first chapter, we will give a presentation about the host institute, an informal introduction to MPG and talk about the state of the art in MPG theory, its relation to other fields, and the existing Machine Learning Learning (ML) methods in related games.

In the second chapter, we will formalise MPG using known results from the existing theory. Then, we will design and implement a MPG library that is based on the theoretical frameworks. This library will be used by default in all the remaining of the project.

In the third chapter, we will generate two datasets of MPG using efficient implementations of graph generation. We will then try to solve all instances of both datasets using our most efficient implementation of MPG solver based on Constraint Satisfaction Problem (CSP) methods. We will use also a High Power Computing (HPC) cluster to further minimise the execution time.

In the fourth chapter, we will analyse briefly both datasets with their annotations, and try to confirm empirically our hypotheses about MPG. We will then design a GNN model that conforms to some desired properties, based in our findings.

In the fifth and final chapter, we will implement a whole distributed RL system that uses Self Play (SP) that refines the learnable parameters of the model discussed in chapter 4. We will base the full architecture of the system in the famous Alpha Zero paper [45]. Finally, we will deploy the system in a large HPC cluster and execute it for refining the model.

1

Project context

Introduction

In this chapter, we are going to present the host department and their research interests.

Then, we will present the project in general and its importance. Later, we will give the functional requirements and non-functional requirements of our implementations.

Furthermore, we will introduce informally Mean Payoff Game (MPG), then dive into the state of the art in its theory and in Machine Learning Learning (ML) approaches. Also, we will briefly introduce learning methods.

Finally, we will present our work methodologies, and describe the whole project schedule in figure 1.10.

1 Presentation of the host institute

1.1 Presentation of the institute of Algebra

Algebra (from Arabic: al-dschabir "the joining of broken parts") is one of the oldest scientific disciplines of all. As a doctrine of solving equations and systems of equations, it already developed in Babylon and in ancient Egypt. In the 19th century this theory ("classical algebra") was largely completed by the fundamental theorem of algebra by Gauss and the theorem by Abel-Ruffini. Modern algebra developed - on the basis of the work of Galois and Abel - as the theory of algebraic structures, above all of groups, rings and solids. Algebra has fundamental importance within mathematics, since almost all mathematics is based on sets and operations or relations on their elements. In addition to mathematics, various sub-areas of algebra are essential for other research areas, e.g. for symmetry studies in physics and chemistry or for coding theory and cryptography in computer science.

Algebra is a rich field of science with many exciting and dynamic fields of research.

1.2 Research interest

The main research areas at the Institute for Algebra are constraint satisfaction problems, finite group theory, and valued rings and fields. All of these topics require the linking of a variety of mathematical theories such as graph theory, group theory, general algebra, order theory and topology.

2 Project definition

2.1 General frame

With the major boom in Artificial Intelligence, and especially Deep Learning in the 21st century. Research institutions are looking for Machine Learning Learning techniques to approximate solutions of hard games that are intractable to humans and even conventional algorithms.

Mean Payoff Game are a class of games on graphs that can be explained very easily¹ to a layman. In the other hand, it is hard even for computers to guess a good strategy, let alone calculate the optimal one. Therefore, a sophisticated ML algorithm is needed for that purpose.

2.2 Motivations

Recent breakthroughs in Reinforcement Learning techniques made superhuman performances in previously intractable games such as Go [44].

Furthermore, for a popular game like Chess, it even surpassed the strongest conventional engines² [45].

This advances were gradually applied to theoretical games such as Stochastic Parity Games [7].

Now, This interesting, as the class of Stochastic Parity Games is a subclass of Stochastic Mean Payoff Game (SMPG), which is our ultimate goal.

Now, to not dive directly into the stochastic version which is more subtle, we limit our approach to Mean Payoff Game (MPG), but try to make it as general as possible, so that it can be incorporated to the stochastic case.

2.3 Presentation of the project

This work is a different approach for solving MPG. Exact methods are currently computationally inefficient, and we will instead opt for approximation methods based on DL.

In this project, we:

¹With a slight modification of rules.

²This was at the time of writing the article in 2017. Now, chess engines are a lot stronger, and incorporated DL methods.

- Formalised and analysed MPG.
- Made a Python library for MPG with time-critical functions implemented in C++.
- Generated and annotated a large dataset of MPG, using a High Power Computing (HPC) system.
- Analysed the results, to empirically verify our unproven hypotheses about the game.
- Conceptualized and implemented a Graph Neural Network (GNN) model that predicts the optimal strategy for a player.
- Implemented a Self Play (SP) algorithm, based on Alpha Zero [45] that uses the GNN model.

,

3 Requirement

3.1 Functional Requirements

The ultimate goal of this project is to design and implement a SP system for MPG.

As we were not able to discover any implementation regarding MPG. Before diving into the modeling part, we implemented and tested a library that we called **MPG**. We generated and annotated two large datasets of MPG to understand more properties about the game and strategies. And finally then, we started implementing a GNN model that will be used as part of the SP system.

3.2 Non-functional requirements

Scope	Requirements
Library	<ul style="list-style-type: none"> Performance: Time-critical operations should have minimal overhead. Correctness: The implemented methods must be formally correct. Modularity: The library should be modular and extensible.
Solver	<ul style="list-style-type: none"> Performance: Solving a MPG should take as less time as possible. Robustness: Each thread should account for a wide range of errors, and resume operation if affected without disrupting the whole program.
Analysis	<ul style="list-style-type: none"> Rigour: The analysis should be as formal as possible, with proofs if possible. Graphical: The analysis should contain graphical visualisations.
Model	<ul style="list-style-type: none"> Dynamic: The model should work on any graph no matter its size. Symmetry: The model should account for the symmetries of the game. Stability: The model's result should only differ slightly when there is a small change in the game parameters. Performance: The model should not be slower than the conventional solver.
Self Play	<ul style="list-style-type: none"> Just in time: The self play algorithm must support just in time generated graphs. Scalability: The self playing system should scale horizontally and vertically. Adaptability: The system can detect new or defect nodes and act accordingly. Continuity: The learning process must be Integration: The model should account for the symmetries of the game.

Table 1.1: Non functional requirements

4 Introduction to Mean Payoff Games

4.1 Mean Payoff Games

A Mean Payoff Game (MPG) is a two-player turn-based game played in a weighted graph on which every player will select an edge from the position of the current token, and then move the token to an adjacent edge. By doing that, both players will get a common reward w where w is the weight of the selected edge.

The goal of the first player is to maximise the average of the rewards, while the goal of the second player is to minimise it.

Player 1 wins if at a certain moment he can secure a positive average. Player 2 wins if at a certain moment he can secure a negative average.

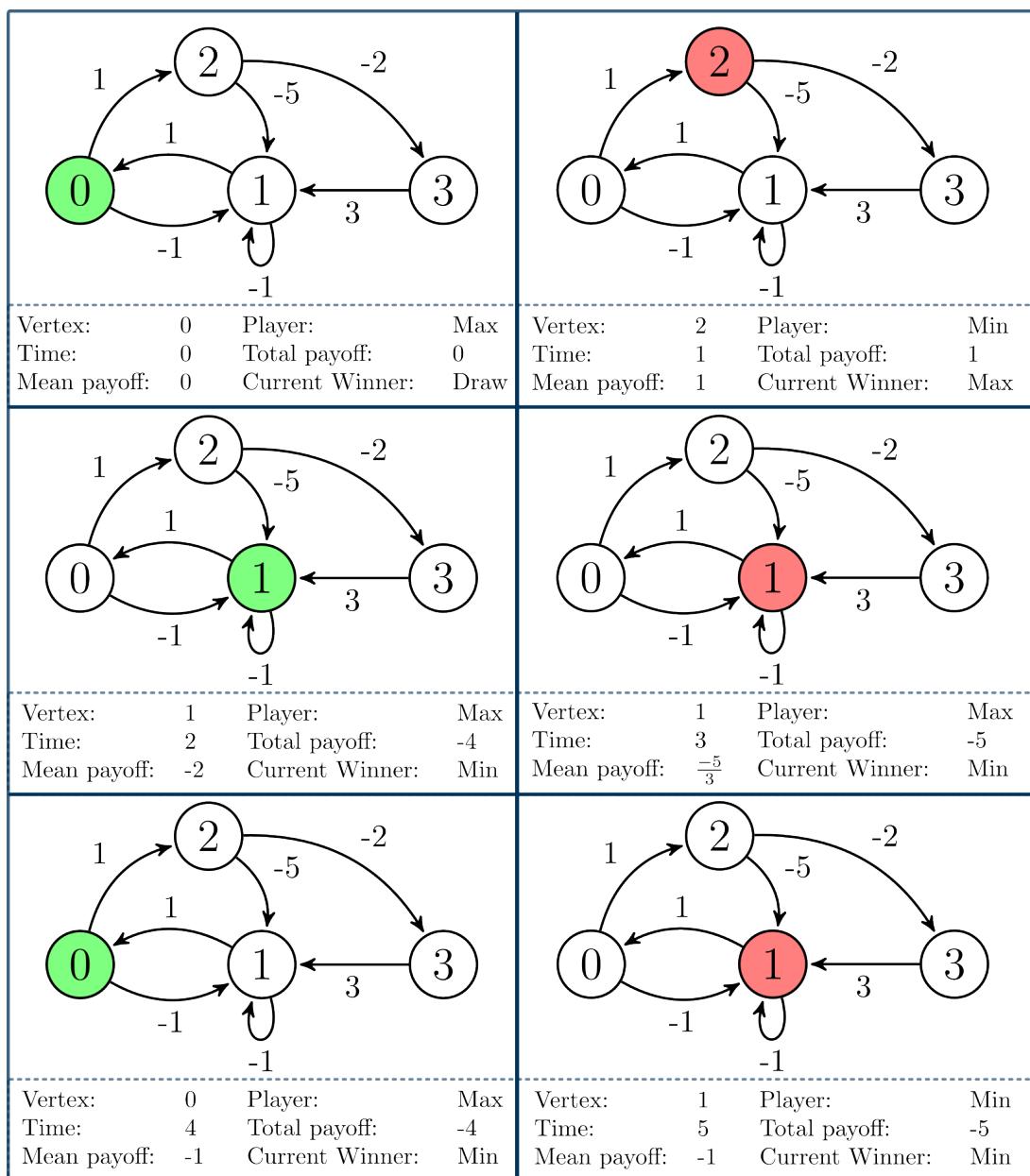


Figure 1.1: A MPG instance

4.2 State of the art

Mean Payoff Game (MPG) are well-known in many fields, such as Optimization [2], Game Theory [14], Formal Verification [19], Constraint Satisfaction Problems [6; 3], Reinforcement Learning [43].

We believe that Ehrenfeucht and Mycielski were the first to introduce MPG in their 1979 paper [14] in which they also proved the optimality of positional strategies³. The problem itself is interesting as it connects many related fields. First of all, it is closely related to many problems in constraint satisfaction [6; 3], model-checking [19], game theory [14].

Also, another interesting fact is that deciding the winner of a mean payoff game is polynomial time equivalent⁴ to the Max Atom problem [3], which is in $\text{NP} \cap \text{co-NP}$, but its membership to P is still open. This is remarkable, there are only few problems that share such fate [36].

This influenced mainly two research axes. The first deals with solving the decision problem⁵, and the other one deals with the optimization problem⁶ related to calculating the optimal strategies. The optimization problem itself can be solved using exact methods [3], as well as iterative methods [43; 2].

While we did not find a Machine Learning Learning (ML) approach on MPG in the literature, we were able to find some results in a related game, known as stochastic parity games. In fact, a model-free Reinforcement Learning (RL) approach was proposed using the Q-learning minimax algorithm [??]. It does only learn specific instances of the game and not a whole family of stochastic parity games.

We were also able to find supervised learning approach on solved instances of that game using Graph Neural Network (GNN) methods. We will base our GNN algorithms.

³Which will be defined in section 1.6.1

⁴Each instance of both problems can be transformed to the other in polynomial time.

⁵The decision problem of a mean payoff game is deciding the winner.

⁶The optimization problem is calculating the best strategy for each player

5 Introduction to learning approaches

5.1 Machine learning

Machine Learning Learning (ML) is a subfield of Artificial Intelligence (AI), which is broadly defined as the capability of a machine to imitate intelligent human behavior. AI systems are used to perform complex tasks in a way that is similar to how humans solve problems.

ML focalises on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

To formalise the definition, we will directly use Mitchell's famous definition [35, page. 2]: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."*

5.2 Deep learning

5.2.1 Neural Networks

Neural Network (NN), are a subset of Machine Learning Learning (ML) and are at the heart of Deep Learning (DL) algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

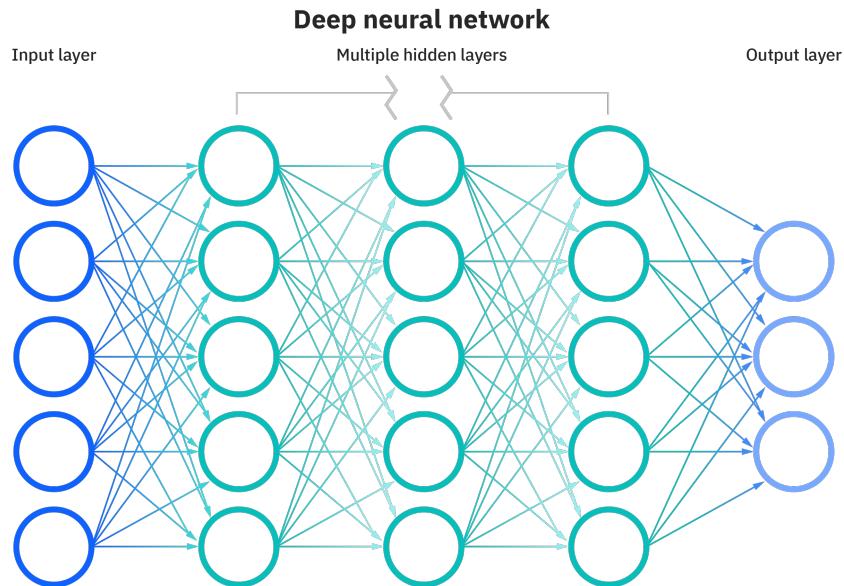


Figure 1.2: An illustration of a neural network

NN are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. These series of connections make the possibility to learn complex features from the dataset.

5.2.2 Definition

Deep Learning (DL) is a subset of Machine Learning Learning (ML), which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—allowing it to “learn” from large amounts of data.

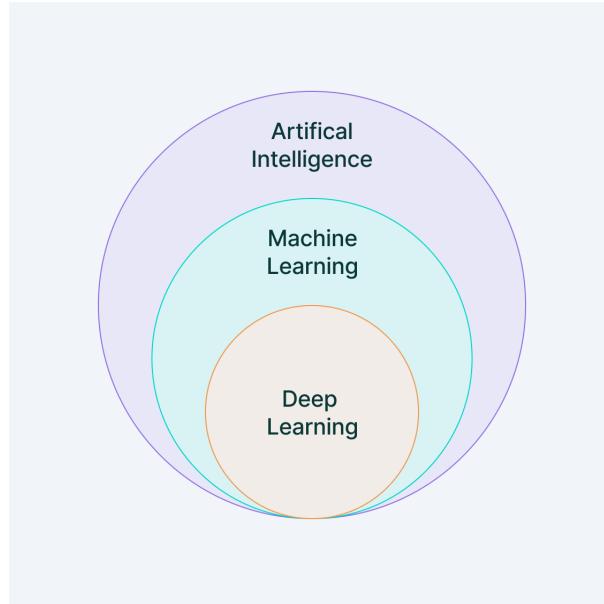


Figure 1.3: Relation between AI, ML and DL

The term deep learning originated from new methods and strategies designed to generate these deep hierarchies of non-linear features by overcoming the problems with vanishing gradients so that we can train architectures with dozens of layers.

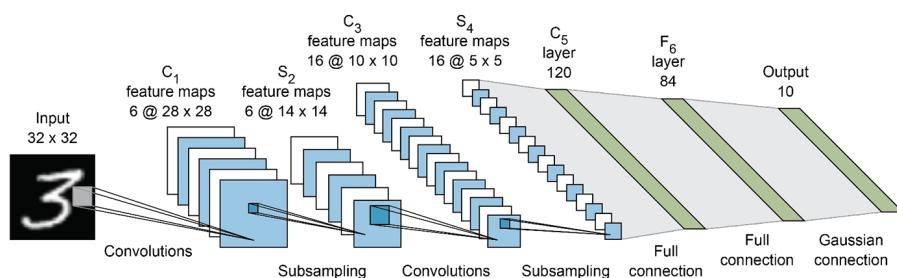


Figure 1.4: An example of deep learning model

5.3 Reinforcement learning

Reinforcement Learning (RL) is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward. This optimal behavior is learned through interactions with the environment and observations of how it responds, similar to children exploring the world around them and learning the actions that help them achieve a goal.

In the absence of a supervisor, the learner must independently discover the sequence of actions that maximize the reward. This discovery process is akin to a trial-and-error

search. The quality of actions is measured by not just the immediate reward they return, but also the delayed reward they might fetch. As it can learn the actions that result in eventual success in an unseen environment without the help of a supervisor, reinforcement learning is a very powerful tool.

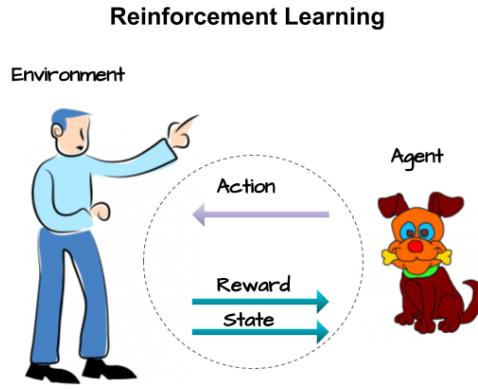


Figure 1.5: An example of a reinforcement learning system

Due to its generality, reinforcement learning is studied in many disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, and statistics.

5.4 Self Play

In Reinforcement Learning (RL), the term self-play describes a kind of Multi-Agent Learning (MAL) that deploys an algorithm against copies of itself to test compatibility in various stochastic environments.

Self Play (SP) rose to prominence after the great success of Alpha Go [44] and its subsequent version Alpha Zero [45].

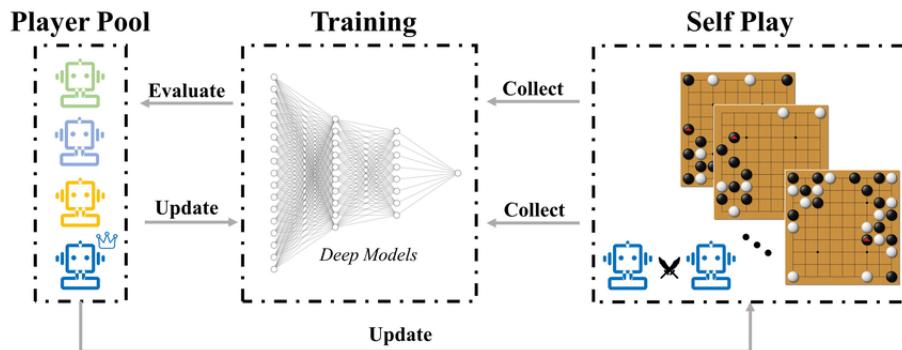


Figure 1.6: Alpha Go pipeline
src:

6 Introduction to high power computing

6.1 HPC

The High Power Computing and Storage Complex (HRSK-II) with about 60,000 CPU cores, designed and operated by ZIH, achieves a peak performance of more than 1.5 quadrillion floating point operations per second (PFLOPS). Massively parallel applications can also leverage the highly efficient processing power of our 448 GPU accelerators. Additionally, the system is equipped with a shared storage consisting of about 2000 high capacity disks and a network of fast solid-state drives for specialized tasks. Therefore it provides the basis to successfully investigate compute-intensive and data-intensive problems of various scientific disciplines, like computational fluid dynamics, weather and climate modeling, material science, electrodynamics, life sciences, and bioinformatics.

6.2 Prerequisites

The ZIH HPC system is Linux based system. To work on the HPC system. Therefore, a HPC user needs to know basic terminology such as SSH, cluster, login node, compute node, local and shared filesystem, command line or shell.

6.3 Architecture

The HPC system is composed of many clusters:

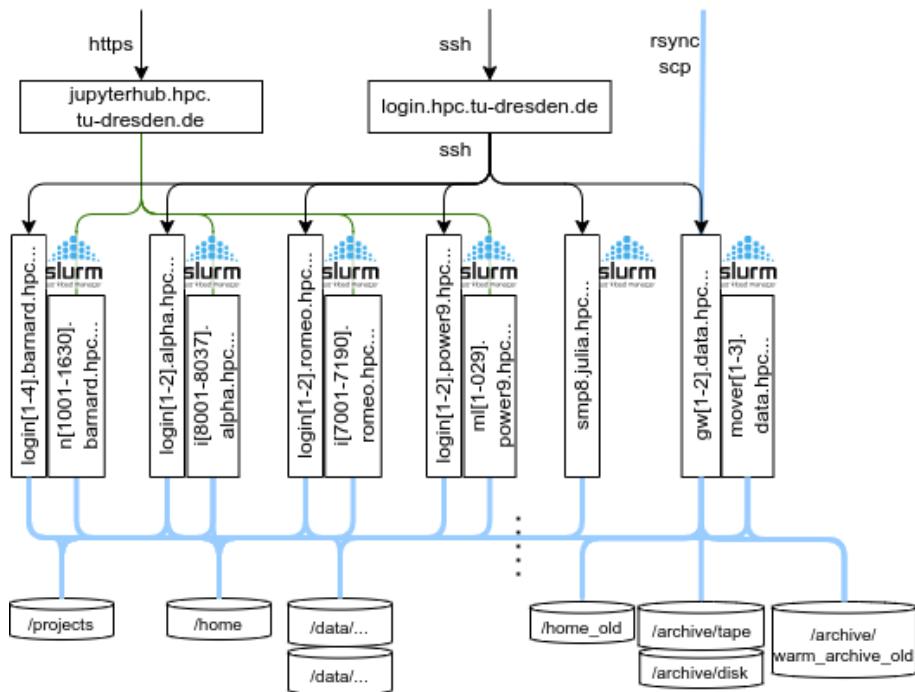


Figure 1.7: HPC architecture

7 Work methodologies

This section is dedicated to describing the three methodologies we selected to conduct this project, complemented by the Gantt chart which illustrates the distribution of our tasks over the duration of the internship.

Each of the three methodologies we selected dealt with a different part of the project. These parts were very different in nature. As a result, we effectively considered them as distinct subprojects.

The choice of more than one methodology was necessary in our case, as each subproject had to be approached with a unique mindset.

7.1 Agile Methodology

As the first goal of the project was to implement and test a MPG library, we followed the **Agile** methodology.

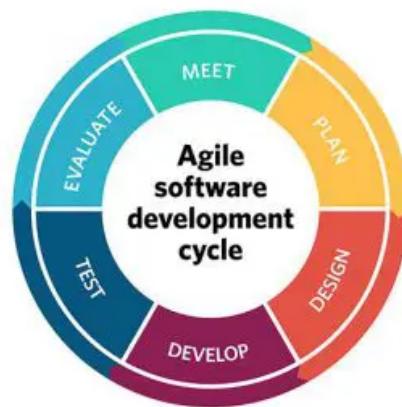


Figure 1.8: Agile development cycle

Agile methodology is a project management approach that prioritizes cross-functional collaboration and continuous improvement. It divides projects into smaller phases and guides teams through cycles of planning, execution, and evaluation.

Agile's entire framework revolves around the program's core values. Many of the Agile Principles are directly based on these values:

7.2 Crisp-DM

The second subproject was to analyse, and implement a reinforcement learning model for MPG. For that part, we used the **Crisp-DM**

CRISP-DM, which stands for Cross-Industry Standard Process for Data Mining, is an industry-proven way to guide the data mining efforts.

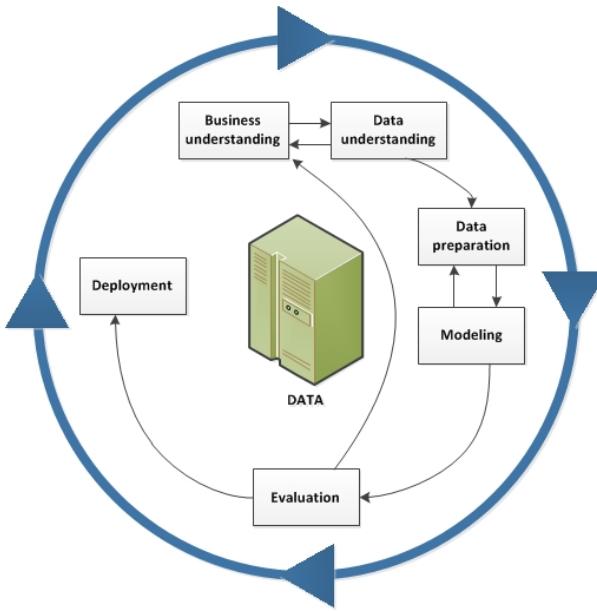


Figure 1.9: Agile development cycle

As a methodology, it includes descriptions of the typical phases of a project, the tasks involved with each phase, and an explanation of the relationships between these tasks. As a process model, CRISP-DM provides an overview of the data mining life cycle.

GANTT Diagram

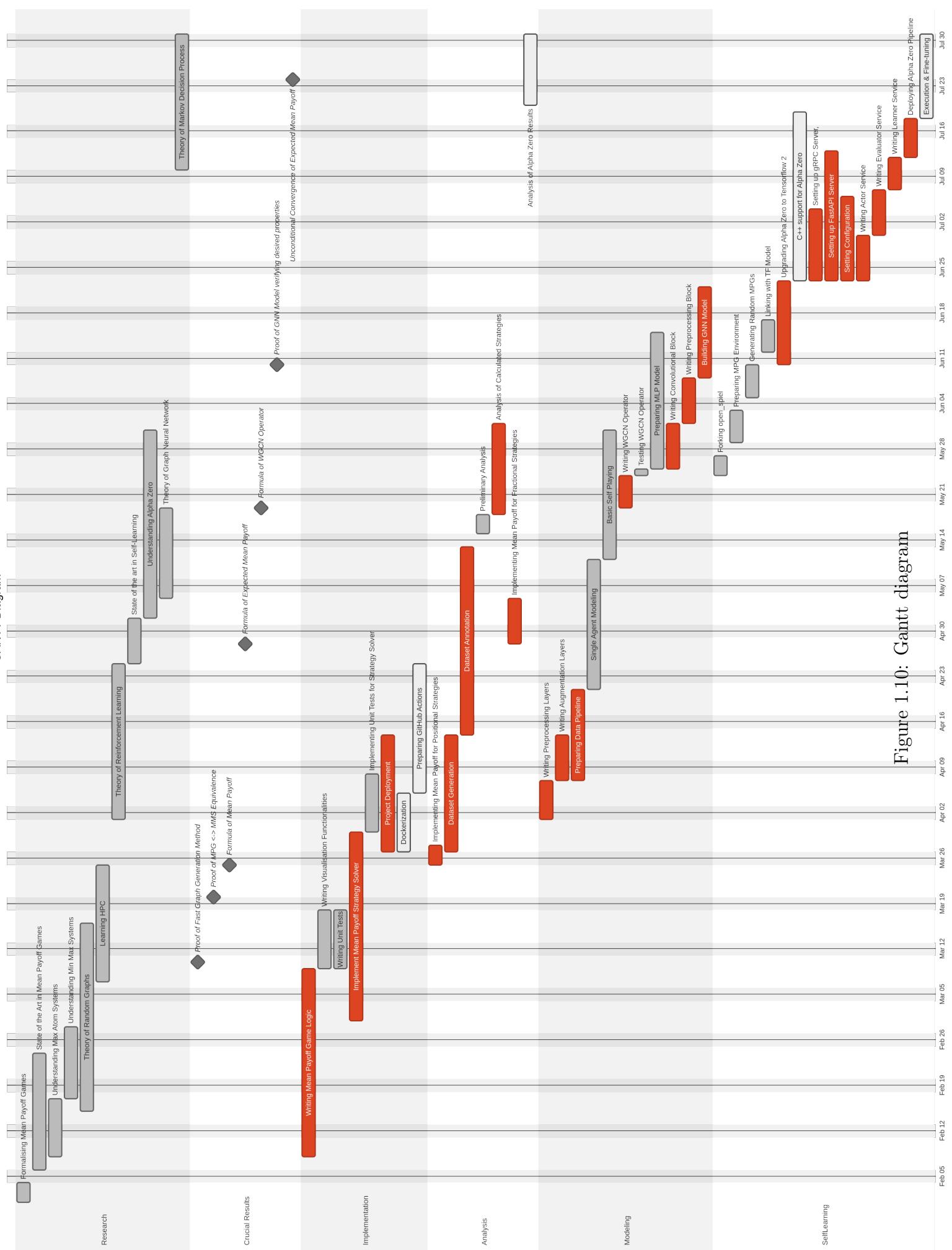


Figure 1.10: Gantt diagram

Conclusion

Through this chapter, we presented the host department and discovered their research interests.

Then, we presented the project and described the functional and non-functional requirements

Later, we introduced MPG and described the state of the art. We then introduced briefly learning approaches.

Finally, we showed our work methodologies, and the detailed work in the scope of this project.

2

Formalisation & Implementation

Introduction

In this chapter, we are going to formalise the definition of a MPG. We are also going to specify in what sense a strategy is optimal.

Later, we will unify the MPG theory frameworks under a modular and extensible library.

1 Formalisation

To define a Mean Payoff Game, we will start by formalising a weighted di-graph¹.

1.1 Di-Graph

A Weighted Di-Graph G is a tuple (V, E, W) where:

- V is the set of vertices.
- $E \subseteq V \times V$ is the set of edges.
- $W : E \rightarrow \mathbb{G}$ is the weight function, assigning a weight for every edge, with \mathbb{G} some ordered abelian group².

¹Directed Graph

²This 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.

1.2 Mean Payoff Game

Formally, a **Mean Payoff Graph** is a tuple (V, E, W, P, s, p) where:

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

A **Mean Payoff Game (MPG)** is a perfect information, zero-sum, turn based game played indefinitely 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	$\liminf_{n \in \mathbb{N}^*} \frac{1}{n} \sum_{k=0}^{n-1} w_k \geq 0$	$\liminf_{n \in \mathbb{N}^*} \frac{1}{n} \sum_{k=0}^{n-1} w_k < 0$	
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	$\liminf_{n \in \mathbb{N}^*} \frac{1}{n} \sum_{k=0}^{n-1} w_k > 0$	$\limsup_{n \in \mathbb{N}^*} \frac{1}{n} \sum_{k=0}^{n-1} w_k < 0$	$\begin{cases} \liminf_{n \in \mathbb{N}^*} \frac{1}{n} \sum_{k=0}^{n-1} w_k \leq 0 \\ \limsup_{n \in \mathbb{N}^*} \frac{1}{n} \sum_{k=0}^{n-1} w_k \geq 0 \end{cases}$

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 modification of C_1 that introduces the possibility of drawing.
- C_3 is symmetric³, and will be used for our machine learning. It was referenced in [6].

Now, there is a slight difference between the three winning conditions.

³It does not give an advantage towards any player.

For example, an optimal strategy with respect to C_1 may not be optimal with respect to C_2 . While this can happen, it is unlikely.

What about C_2 and C_3 ? As different as they appear, they are equivalent in the scope of this report⁴. This is demonstrated in 1.7.

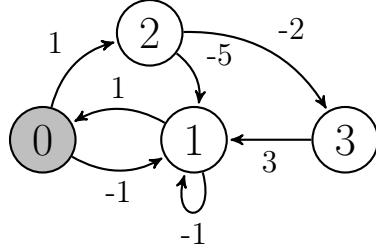


Figure 2.1: An example of a MPG

1.3 Well Founness

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:

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

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 in the table 2.1 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 MPG instance:

⁴They are still different conditions in general.

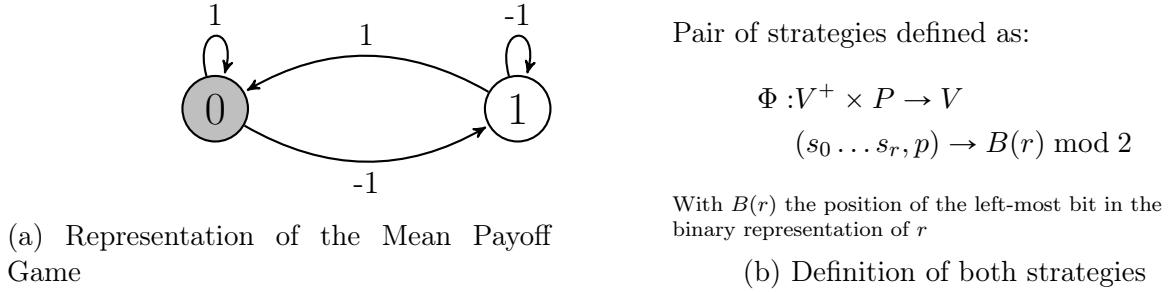


Figure 2.2: An example of an execution with non-convergent Mean Payoffs

1.4 Properties

A MPG has many properties that are interesting from a game theory perspective.

1.4.1 Two Player

The game is a two player game.

1.4.2 Turn Based

1.5 Symmetries

Mean Payoff Games exhibits many natural symmetries.

1.5.1 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.

1.6 Strategy

1.6.1 Deterministic Strategies

Let p be a player.

A (deterministic) strategy is a function $\Phi^p : V^+ \rightarrow V$ such that:

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

If the strategy does only depend on the current vertex, we say it is a memoryless (deterministic) strategy.

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 MPG.

Positional strategies are crucial for our analysis as a result of the following theorem [14]:

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

1.6.2 Probabilistic Strategies

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

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

If the probabilistic strategy does only depend on the current vertex, we say it is a memoryless probabilistic strategy. $\Pi : V \rightarrow V$

In this report, we will use both the terms “fractional strategies” and “behavioural strategies” interchangeably as an alias for memoryless probabilistic strategies, which is also conforming to the established litterature of MPG.

1.6.3 Considered Strategies

Strategies that depends in complete past histories are in general intractable. For a MPG, it is proven that we always have an optimal **positional** strategy [14].

For that we will only consider **memoryless** strategies. First of all, it is natural to consider positional strategies thanks to theorem 2. For the fractional strategies, they serve as a generalisation of positional strategies that can be used in Machine Learning Learning (ML) methods, as they reside in a smooth space.

1.6.4 Positional Optimal Strategy

There are three kinds of optimality:

1.6.4.1 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

1.6.4.2 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

1.6.4.3 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 hierarchy considering the set of optimal strategies:

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

1.7 Mean Payoff

We have used the word "Mean Payoff" extensively, and they are the central entity in mean payoff games⁵ 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.

1.7.1 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 follow⁶:

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

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

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:*

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

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

⁵This explains the name "mean payoff game".

⁶For probabilistic strategies, both terms are random variables, so we will be interested in their expected values.

1.7.2 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 2.2.

Now, fortunately, this is not an issue, as we are interested in good strategies. Theorem 2 proven in [14] 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 problem⁷ to the more tractable positional⁸ and behavioural⁹ 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.*

We will give a constructive proof of theorem 4 in section 2. 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.

1.7.3 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:

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

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

2 Evaluating Strategies

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

⁷The problem of finding optimal strategies

⁸memoryless and deterministic

⁹memoryless and probabilistic

2.1 Monte-Carlo Simulations

This is the most intuitive evaluation method

2.2 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:

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

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.:

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

Now it can be proven that:

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

With that:

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

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

- Calculating the transition function $F : V \times P \rightarrow V \times P$. 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.

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

```

 $F \leftarrow \text{Transition}(G, \Phi^{\text{Max}}, \Phi^{\text{Min}})$                                  $\triangleright$  Calculate the transition function
 $x_0 \leftarrow (s, p)$ 
 $(T, r) \leftarrow \text{FloydCycleFinding}(F, x_0)$                                  $\triangleright$  Find the period and the offset
 $S \leftarrow 0$                                  $\triangleright S$  represents the cumulative payoffs along a cycle
 $x \leftarrow x_0$ 
for  $k \in \{1, \dots, r\}$  do                                 $\triangleright$  Advance until arriving to the cycle
     $x \leftarrow F(x)$ 
end for
for  $k \in \{1, \dots, T\}$  do
     $y \leftarrow F(x)$ 
     $u \leftarrow \text{projection}(x)$                                  $\triangleright$  Extracts the current vertex
     $v \leftarrow \text{projection}(y)$                                  $\triangleright$  Extracts the next vertex
     $V \times P \rightarrow V$ 
     $V \times P \rightarrow V$ 
     $S \leftarrow S + W(u, v)$ 
     $x \leftarrow y$ 
end for
return  $R \leftarrow \frac{S}{T}$ 

```

2.3 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

$(A, W) \leftarrow \text{MRP}(G, \Phi^{\text{Max}}, \Phi^{\text{Min}})$ \triangleright Extract the MRP form. This is detailed in 1.2

$u \leftarrow (A \odot W) \mathbf{1}$

$X \leftarrow \text{NullSpace}(\text{Id} - A)$ \triangleright Extract the kernel-basis of $\text{Id} - A$

$Y \leftarrow \text{NullSpace}(\text{Id} - A^T)$ \triangleright Extract the kernel-basis of $\text{Id} - A^H$.

$T \leftarrow X(Y^T X)^{-1} Y^T$ \triangleright Calculate the limit $\lim_{n \rightarrow +\infty} \frac{1}{n} \sum_{k=0}^{n-1} A^k$. This is detailed in 2.3

return $\mathbb{E}[R] \leftarrow Tu$

3 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 solved 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.

3.1 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:

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

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 $\mathcal{O}(|V|^3)$

3.2 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.

4 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 serialization, visualisation and support for machine learning methods.
- The second one is called **mpgcpp**. It is implemented in C++ to maximize efficiency. Also, the time-critical C++ functions are exported to Python. In they can be used via the **wrapper** module.

4.1 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:

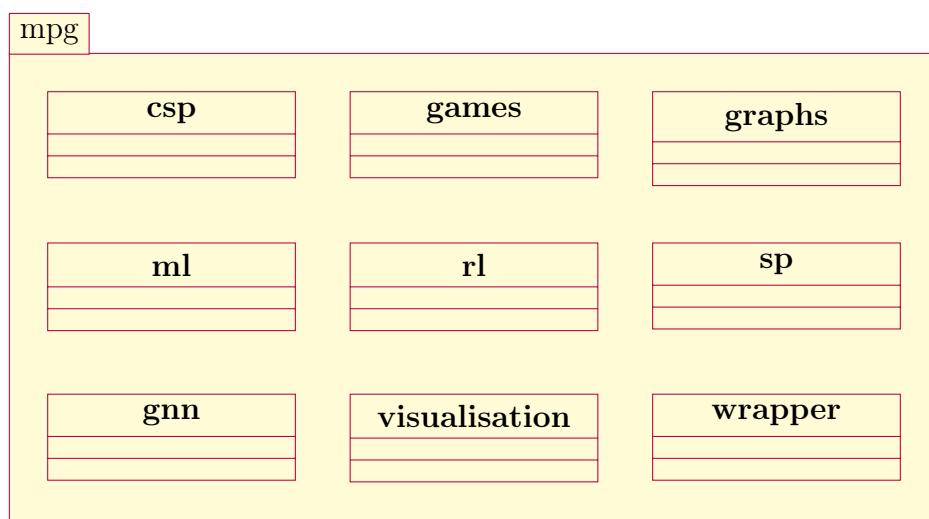


Figure 2.3: **mpg** library

This library contains the following modules:

4.1.1 csp

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

They are described in details in the appendix A.

4.1.2 graphs

This module 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

4.1.3 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.

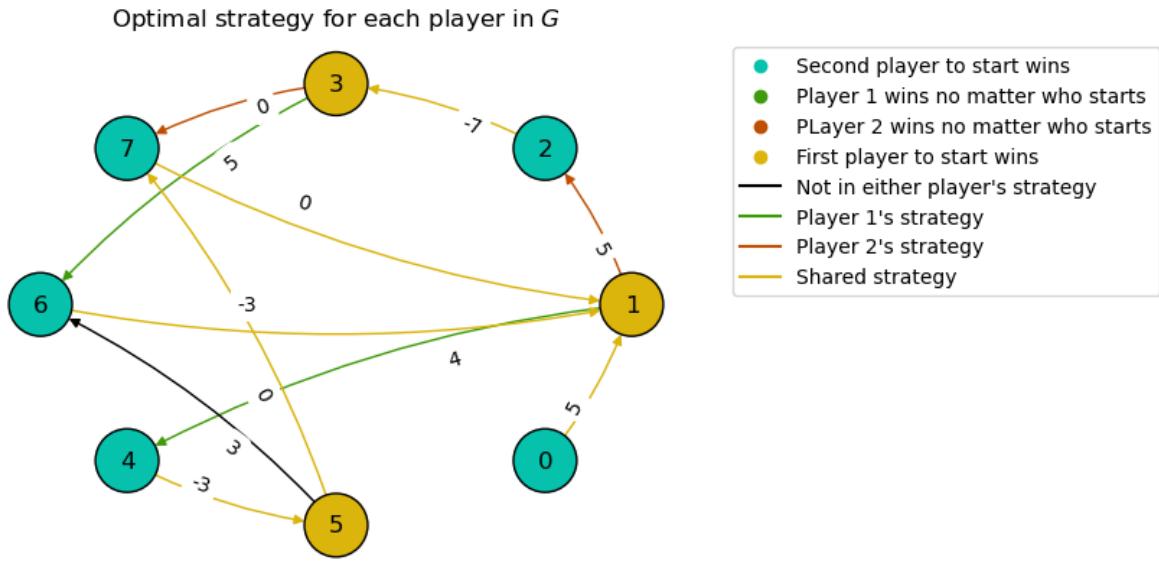


Figure 2.4: Generated visualisation for a Mean Payoff Game with the optimal strategies

4.1.4 games

This module defines the core functionalities related to mean payoff games. It also defines the methods to read/write Mean Payoff Graphs. It uses the weighted edgelist format, and supports compression.

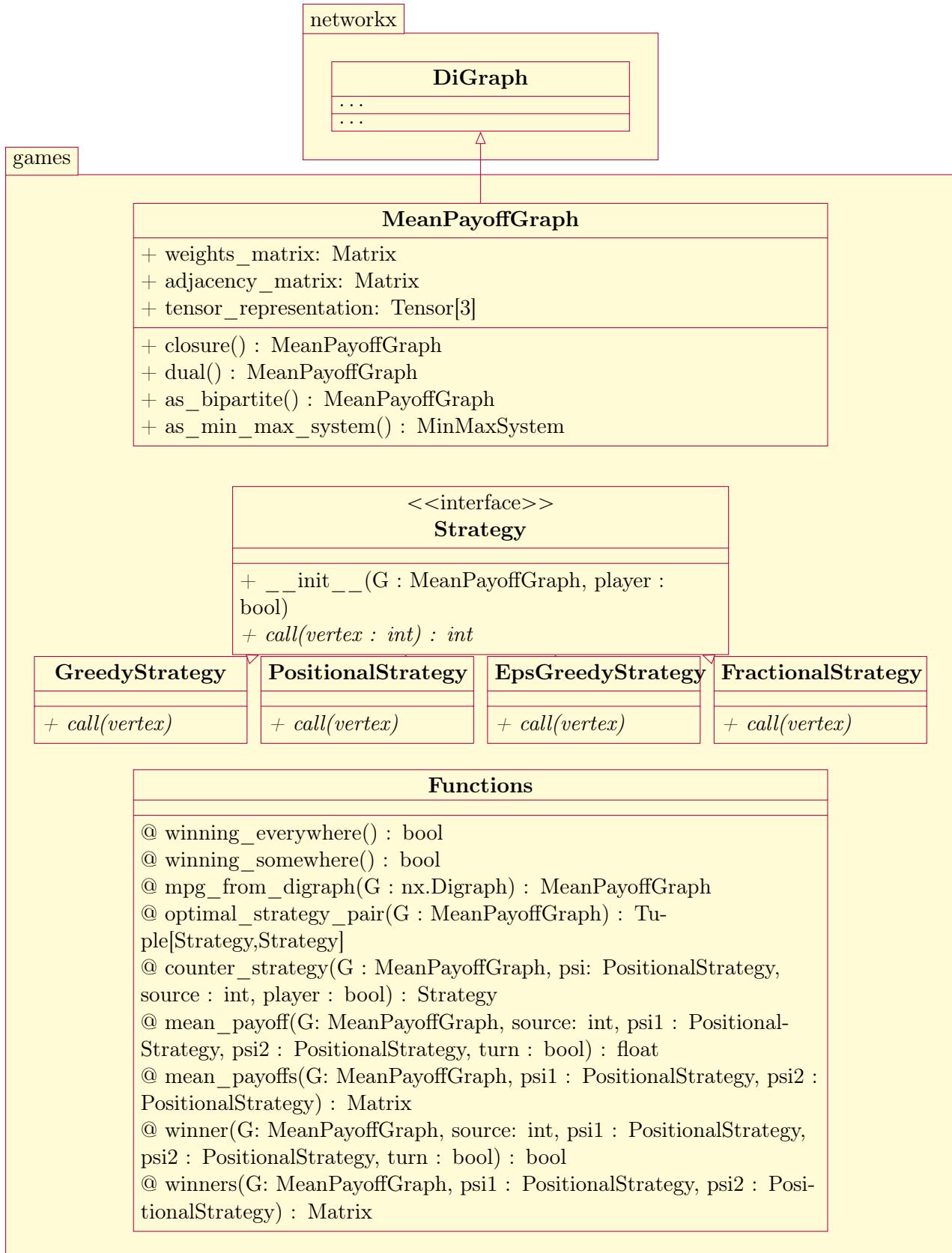


Figure 2.5: **games** module

4.1.5 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.

4.1.6 gnn

This module defines the basic functionalities of graph neural networks [42] that are required for our models.

4.1.7 rl

This module defines the required functions to do reinforcement learning on Mean Payoff Games¹⁰.

4.1.8 sp

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

4.1.9 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.

4.2 mpgcpp

This library contains the

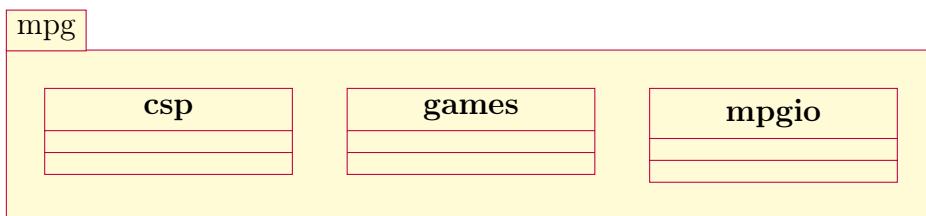


Figure 2.6: **mpgcpp** library

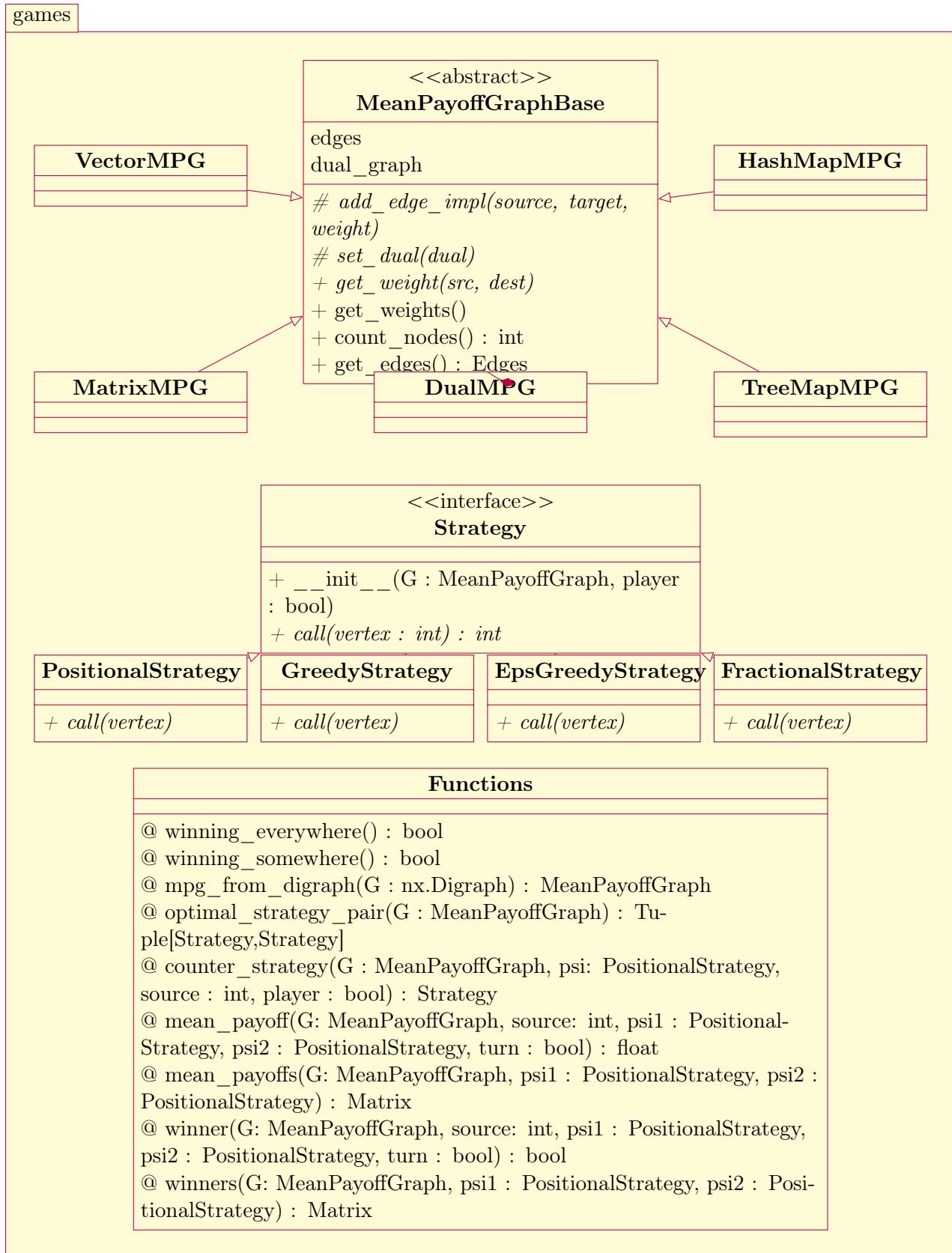
4.2.1 csp

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

4.2.2 games

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

¹⁰The current version of this module only supports Mean Payoff Games where at least one player has a fixed strategy.


 Figure 2.7: **games** module

4.2.3 mpgio

Unlike **mpg**, which has support for mean payoff graph I/O from files via **NetworkX**. In the C++ library, we have to implement this functionalities from scratch. We used standard I/O utilities with **boost** to interact with compressed streams conforming to the format used by **mpg**.

4.3 Environment

4.4 Testing

4.5 Structure

Conclusion

In this chapter, we formally defined MPG, the strategies, and the mean payoff itself. We showed that while it can be problematic to have a mean payoff in the general sense, This will not be an issue in our scope.

Then, we implemented crucial algorithms for countering strategies, and calculating mean payoffs that will serve as an evaluation metric for the future Machine Learning Learning (ML) model.

Finally, we designed an implemented a MPG library on which we will base our work. We implemented time-critical algorithms with little overhead in C++ and linked it to the python implementation. To also test the correctness of our implementations, we added unit tests to each important algorithm..

3

Dataset Generation & Annotation

Introduction

This chapter will be dedicated to the generation of random MPG. We will decompose this problem into generating a graph and generating weights.

Later, we will detail graph notable distributions, and will implement efficient algorithms to build random graphs. Also, we will add a conditionning on the graph distributions to avoid having any sinks.

Furthermore, we will propose some desired properties of MPG distributions, and we will show the graph and weight distributions.

Then, we will detail our steps to generate two MPG datasets, one for sparse graphs, and the other for dense ones. Also, we will annotate both datasets by finding the optimal strategies.

Finally, we will deploy all our implementations to a HPC system to reduce execution time.

1 Decomposition

Generating a MPG can be decomposed into two subsequent objectives.

1. Generate the graph itself.
2. Generate the weights

We will follow this approach to generate a random MPG. In fact, we will decompose MPG distributions into a graph distribution \mathcal{G} and a weight distribution \mathcal{W} .

We will also require that \mathcal{W} is scalar and does not have any information about the generated graph in the sense that building the weights function is done by repeated

sampling from \mathcal{W}

2 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.

We will offer here a directed variant of these two distributions that suits for MPG.

2.1 $\mathcal{D}(n, p)$ Family

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

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

This is the directed version of the binomial random graph distribution $\mathcal{G}(n, p)$ [16, Section. 1.1].

2.2 $\mathcal{D}(n, m)$ 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, \dots, e_m were drawn uniformly and without replacement from the set $V \times V$

This is the directed version of the uniform random graph distribution $\mathcal{G}(n, m)$ [16, Section. 1.1].

While $\mathcal{D}(n, m)$ has a more natural definition, the analysis of $\mathcal{D}(n, p)$ is easier. Also there are many results showing similarities between the two distributions [16, Section. 1.1]. For this reason, we will focus primarily on $\mathcal{D}(n, p)$ graphs.

2.3 $\mathcal{D}(n, p)$ Graph Construction

2.3.1 Naive Method

The definition of $\mathcal{D}(n, p)$ gives a straightforward construction.

This is achieved by flipping a coin¹ 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:

¹The coin is potentially biased with a probability of obtaining head equal to $p \in [0, 1]$

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: $G \sim \mathcal{D}(n, p)$

$A : (u, v) \in V \times V \rightarrow 0$
for $u \in V$ **do**
 for $v \in V$ **do**
 Generate $X \sim \mathcal{B}(p)$ $\triangleright \mathcal{B}(p)$ is the bernoulli distribution
 $A(u, v) \leftarrow X$
 end for
end for
return $G \leftarrow \text{GraphFromAdjacencyMatrix}(A)$

The complexity² of the following algorithm is $\mathcal{O}(n^2)$.

2.3.2 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 distribution³
- 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: $G \sim \mathcal{D}(n, p)$

$A : u \in V \rightarrow \emptyset$
for $u \in V$ **do**
 Generate $d \sim \mathcal{B}(n, p)$ $\triangleright d$ represents the degree, $\mathcal{B}(n, p)$ is the binomial distribution
 $A(u) \leftarrow \text{choice}(V, d)$
end for
return $G \leftarrow \text{GraphFromAdjacencyList}(A)$

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, \dots, d_n is:

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

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

²We assume the cost of generating a Bernoulli random variable as $\mathcal{O}(1)$

³Or the ingoing degree distribution, they are in fact equal.

2.4 Choice Function

2.4.1 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: $m \in \{0, \dots, |S|\}$ the number of chosen elements

Ensure: H a set of size m containing uniformly drawn elements without replacement.

```

 $H \leftarrow \emptyset$ 
while  $|H| < m$  do
    Generate  $v \sim \mathcal{U}(S)$             $\triangleright$  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:

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

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 $H_0 = 0$.

2.4.2 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 is⁴ $\tilde{\mathcal{O}}(m \log m)$.

To prove this result, we use a well-known asymptotic approximation of the Harmonic series⁵ [26, 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 $m = km + o(m)$, $k \in [0, 1[$:

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

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

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

⁴Here we use the minus sign to emphasize that $m \leq n$

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

2.4.3 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.

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  $\frac{m}{|S|} \leq \tau$  then
     $H \leftarrow \text{choice}(V, n)$ 
else
     $H \leftarrow S \setminus \text{choice}(S, n - m)$ 
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 independent⁶ of n . So that the Big-O bound is only a dependent on m .

2.5 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:

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

⁶The 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.

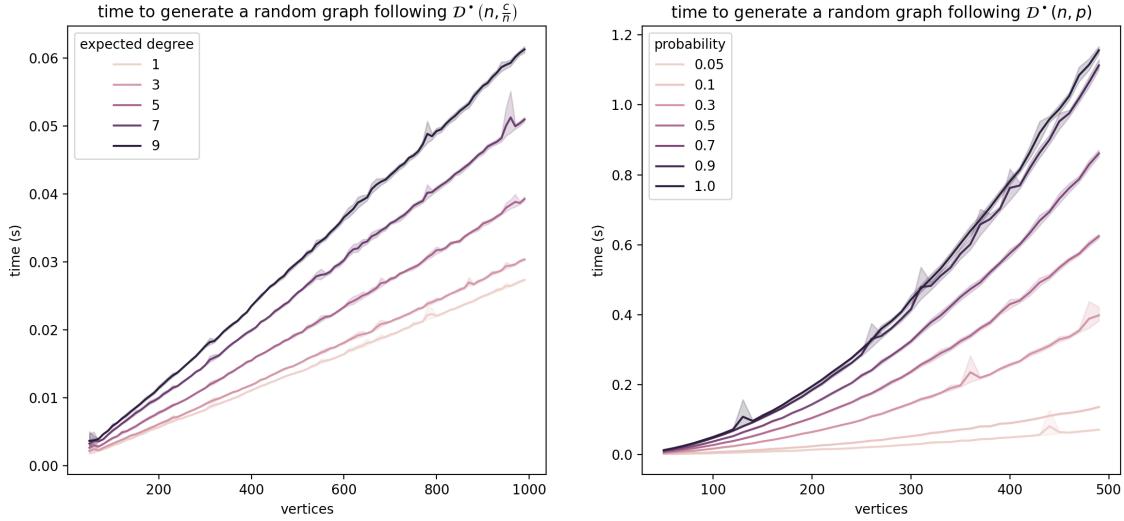


Figure 3.1: Experimental performance to generate dense & sparse graphs

2.6 $\mathcal{D}(n, m)$ 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 purpose⁷:

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: $G \sim \mathcal{D}(n, m)$

$E \leftarrow \text{choice}(\text{Lazy}(V) \times \text{Lazy}(V), m)$ \triangleright We only need the m elements on-demand.

return $G \leftarrow \text{GraphFromEdges}(E)$ \triangleright This justifies using Lazy

Here $\text{Lazy}(V) \times \text{Lazy}(V)$ is a lazy implementation of cartesian product that supports bijective indexing⁸ over $\{0, \dots, n^2 - 1\}$.

The complexity of this construction is: $\tilde{\mathcal{O}}(m)$

⁷It 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.

⁸Indexing is required for uniform sampling

3 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 mitigate this, we will impose a conditionning on both distribution that will gives a guaranteed Mean Payoff Graph.

We will explore such conditionning both distribution:

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

3.1 Graph rejection method

3.1.1 Definition

We define the graph rejection method as the method used to generate a graph by rejecting bad ones until getting the desired one.

In our case, we will reject graphs that have at least a sink, and retry again until getting a sinkless graph.

3.1.2 Estimating Complexity

The number of steps in this method is proportional tp the number of rejections before getting a sinkless graph.

The number of such rejections will follow a geometric law $\mathcal{G}(h(n, p))$ where:

$$h(n, p) = \mathcal{P}(\text{Sinkless}(\mathcal{D}(n, p))) = (1 - (1 - p)^n)^n \quad (3.3)$$

Proof of (3.3) is found in section 1.3 of appendix B.

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

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

3.1.2.1 Dense Graph case

Now it is clear for dense enough graphs, in particular with $p(n) \geq \frac{\ln(n)}{n}$ for large enough n , the expected complexity will reduce to $\tilde{\mathcal{O}}(n + m)$. This is a consequence of equation (B.3).

Thus: **In the dense case, we consider the graph rejection method to be efficient.**

3.1.2.2 Sparse Graph case

If $p(n) = \frac{k \ln n}{n} + c$ with $k < 1$, we have as a result of equation (B.2):

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

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

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

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

Thus: **In the sparse case, we consider the graph rejection method to be inefficient.**

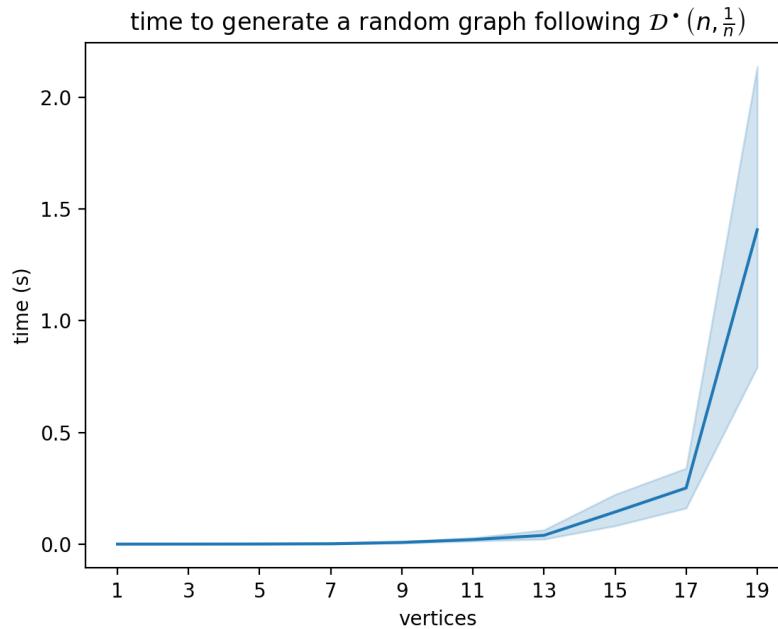


Figure 3.2: Exponential blowup when using the graph rejection method

3.2 Degree rejection method

Instead of throwing the whole graph at once. For every vertex $u \in V$, we try to construct the adjacent vertices of u , and repeat if the procedure gives $\text{Adj } u = \emptyset$.

Algorithm 8 $\mathcal{D}^*(n, p)$ Fast graph generation

Require: $n \in \mathbb{N}^*$ the size of the graph

Require: $p \in \mathbb{N}^*$ the edge probability

Ensure: $G \sim \mathcal{D}(n, p)$

$A : u \in V \rightarrow \emptyset$

for $u \in V$ **do**

repeat

 Generate $d \sim \mathcal{B}(n, p)$ $\triangleright d$ represents the degree, $\mathcal{B}(n, p)$ is the binomial distribution

until $d > 0$

$A(u) \leftarrow \text{choice}(V, d)$

end for

return $G \leftarrow \text{GraphFromAdjacencyList}(A)$

With this trick, the expected complexity will reduce for both algorithms to:

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

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

$$1 - (1-p)^n \geq 1 - (1 - \frac{1}{n})^n \geq 1 - e^{-1}$$

Therefore, the expected complexity will simplify to:

$$\tilde{\mathcal{O}}(n+m)$$

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

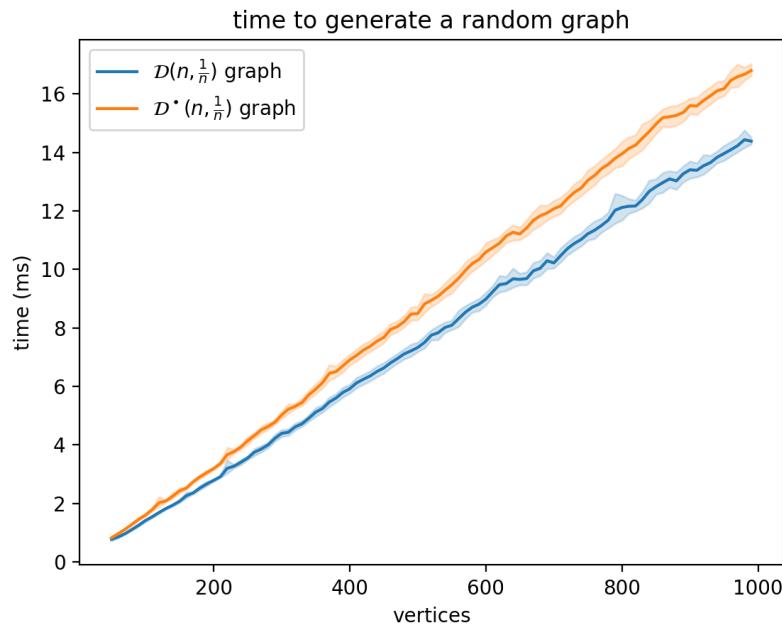


Figure 3.3: Small slowdown due to the degree rejection method

4 Proposed MPG distributions

4.1 Desired properties of MPG distributions

4.1.1 Fairness

This is essential, as we intend to generate a sequence of Mean Payoff Games that do not favour statistically a certain player.

A game distribution is said to be fair if:

$$\mathcal{P}(\text{Max wins } G \text{ assuming optimal strategies}) = \mathcal{P}(\text{Min wins } G \text{ assuming optimal strategies}) \quad (3.4)$$

Furthermore, if we assume C_2 or C_3 as the winning conditions, (3.4) is equivalent to the following equation:

$$\mathcal{P}(v(G) > 0) = \mathcal{P}(v(G) < 0)$$

4.1.2 Symmetric

A real distribution is said to be symmetric if:

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

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.

We will require a symmetric MPG as we do not want a player to have an inherit advantage over the other one⁹

4.2 Implemented Distributions

The following table resumes the implemented distributions:

⁹Other than the first move.

Distribution Family	Parameters	Type
$\mathcal{D}(n, p)$	<ul style="list-style-type: none"> • n : Graph size • p : Edge probability 	Graph distribution
$\mathcal{D}(n, m)$	<ul style="list-style-type: none"> • n : Graph size • m : Number of edges 	Graph distribution
$\mathcal{U}_{\text{discrete}}(-r, r)$	<ul style="list-style-type: none"> • r : The radius of the support 	Weight distribution
$\mathcal{U}(-r, r)$	<ul style="list-style-type: none"> • r : The radius of the support 	Weight distribution
$\mathcal{N}(0, \sigma)$	<ul style="list-style-type: none"> • 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.

5 MPG Generation

5.1 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

5.1.1 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 $G_1^{n,p}, \dots, G_K^{n,p} \sim \mathcal{G}(n, p)$

The total number of examples is:

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

5.1.2 Sparse Graphs

5.2 Implementation

We have implemented a python application called

5.3 Deployment

We have deployed the pipeline in the HPC system. We have launched the following command.

The figure below shows the pipeline:

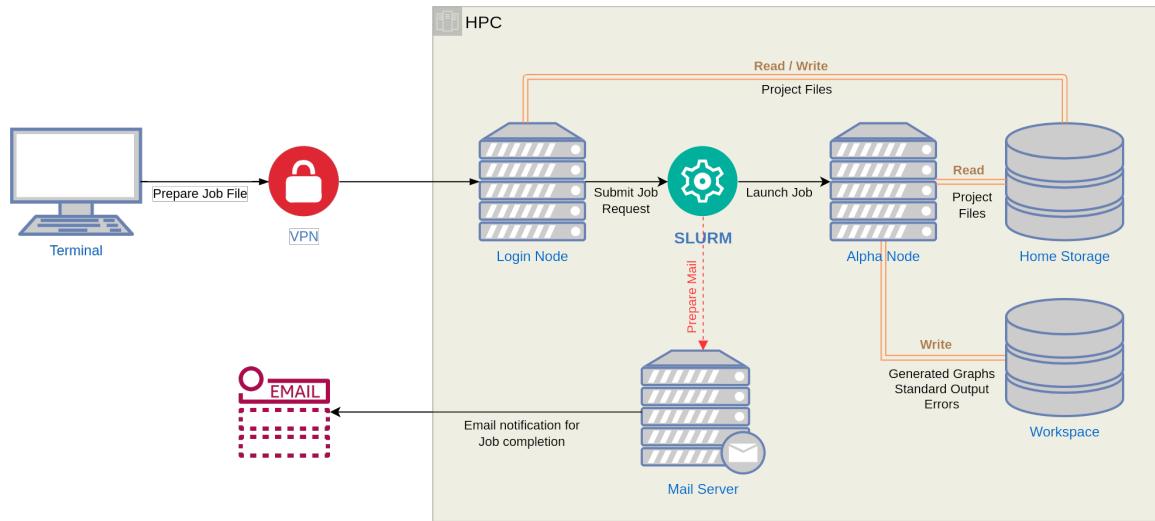


Figure 3.4: HPC pipeline to generate mean payoff graphs

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

6 Annotation

6.1 Approach

We used the Constraint Satisfaction Problem (CSP) algorithm 22 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.

6.2 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 not¹⁰. Equal to "OK" if the execution is successful.
- **running_time**: The time needed to solve the instance.

6.3 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.

6.3.1 Linear Bound

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 $\|W\|_\infty$

$$\Delta X = \sup_{u \in V} X(u) - \inf_{u \in V, X(u) > -\infty} X(u) = \mathcal{O}(\|W\|_\infty)$$

¹⁰We 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. 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 a family of ternary max atom systems where the $\|W\|_1$ estimation is tight. This is demonstrated in the following figure:

$$\begin{cases} X_0 & \leq \max(X_0, X_1) - 1 \\ \dots & \dots \\ X_{n-1} & \leq \max(X_{n-1}, X_n) - 1 \end{cases}$$

Figure 3.5: A counter example to the Linear Bound heuristic

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

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

6.3.2 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 $\nu(t) \leftarrow -\infty, \forall t$

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 $\nu : V \rightarrow \mathbb{Z}$, $X + t$ 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 analogous 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.

6.4 Implementation

6.4.1 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 problem¹¹ for Mean Payoff Games.

It works by 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.

¹¹In 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: $\Phi : V \times P \rightarrow V$ The optimal strategy of each player

```

 $\Phi \leftarrow \emptyset$ 
for  $p \in P$  do
    if  $p$  is Max then
         $G' \leftarrow G$ 
    else
         $G' \leftarrow \bar{G}$ 
    end if
     $S \leftarrow \underset{\text{MPG} \rightarrow \text{Min-Max}}{\text{transform}}(G', D)$ 
     $S' \leftarrow \underset{\text{Min-Max} \rightarrow \text{Max}}{\text{transform}}(S)$ 
     $S'' \leftarrow \underset{\text{Max} \rightarrow \text{Max}_3}{\text{transform}}(S')$ 
     $L \leftarrow \inf D$ 
     $R \leftarrow \sup D$ 
     $Q \leftarrow \emptyset$ 
     $\mathcal{V} \leftarrow \text{Variables}(S'')$ 
    for  $u \in \mathcal{V}$  do
         $X(u) \leftarrow R$ 
        append( $Q, u$ )
    end for
    while  $Q \neq \emptyset$  do
         $X \leftarrow \text{arcconsistency}(S'', X, Q, L)$ 
    end while
    for  $\mathcal{C} \in S''$  do                                 $\triangleright$  Iterate over constraints of  $S''$ 
         $\text{OP} \leftarrow \text{Operator}(\mathcal{C})$            $\triangleright$  Get the operator of  $\mathcal{C}$ . Either Max or Min
         $Y$  the right-hand side variables of  $\mathcal{C}$ 
         $C$  the right-hand side constants of  $\mathcal{C}$ 
         $x$  the left-hand side variable of  $\mathcal{C}$ 
         $u \leftarrow \underset{V \times P \rightarrow V}{\text{projection}}(x)$            $\triangleright$  Extract the vertex
        if  $\text{OP}$  is Max then
             $y^*, c^* \leftarrow \underset{(y,c) \in \text{zip}(Y,C)}{\text{argmax}} \{X(y) + c\}$        $\triangleright$  Extracts the maximum assignment
             $\Phi(u, p) \leftarrow \underset{V \times P \rightarrow V}{\text{projection}}(y^*)$            $\triangleright$  The Strategy is the vertex of the maximum
        assignment
        end if
    end for
end for
return  $\Phi$ 

```

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

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.

6.4.2 Complexity Analysis

For simplicity will suppose that the domain D is finite¹², our algorithm runs in:

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

Otherwise, if D is a real domain, the algorithm still converges if D is bounded¹³, 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 5, we recall that their weights are finite.

7 Deployment

7.1 Pipeline

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 threads working simultaneously on a partition of the dataset.

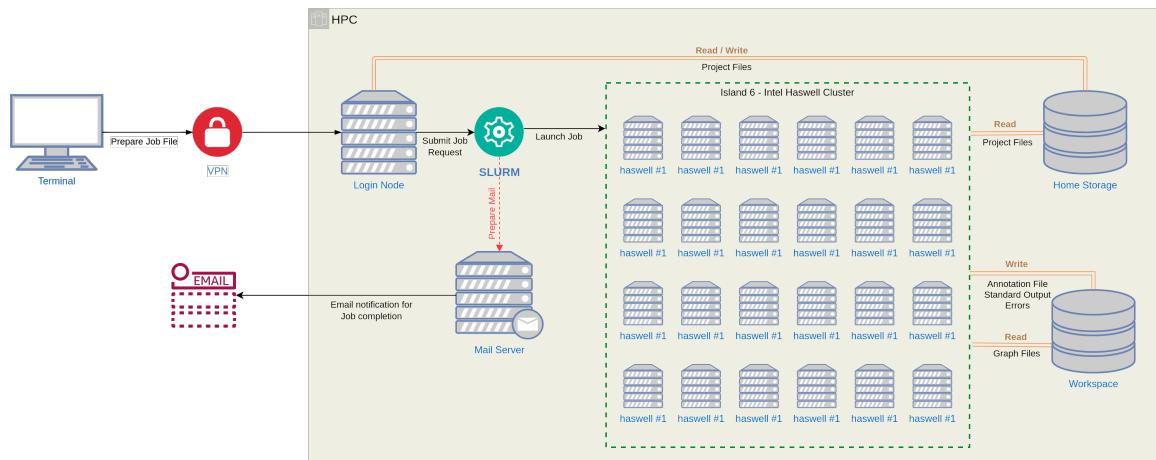


Figure 3.6: HPC pipeline to annotate mean payoff graphs

¹²In particular, a finite subset of \mathbb{Z}

¹³ $D \subseteq [a, b]$ for some interval $[a, b]$

The figure 3.6 above highlights the pipeline used to annotate mean payoff graphs. For illustrative purposes, we omitted the following details:

- The SLURM manager splits the graph files and partition them along the computing nodes.
- Each node produces its own annotation file, which is **unique** per computing node. So we do not have any race condition.¹⁴
- Each of the standard output and error streams is transferred to a file, which is **unique** per computing node.
- Once all jobs terminate, we have a helper script that concatenates all the results into a single file.

¹⁴Processes that access the same file.

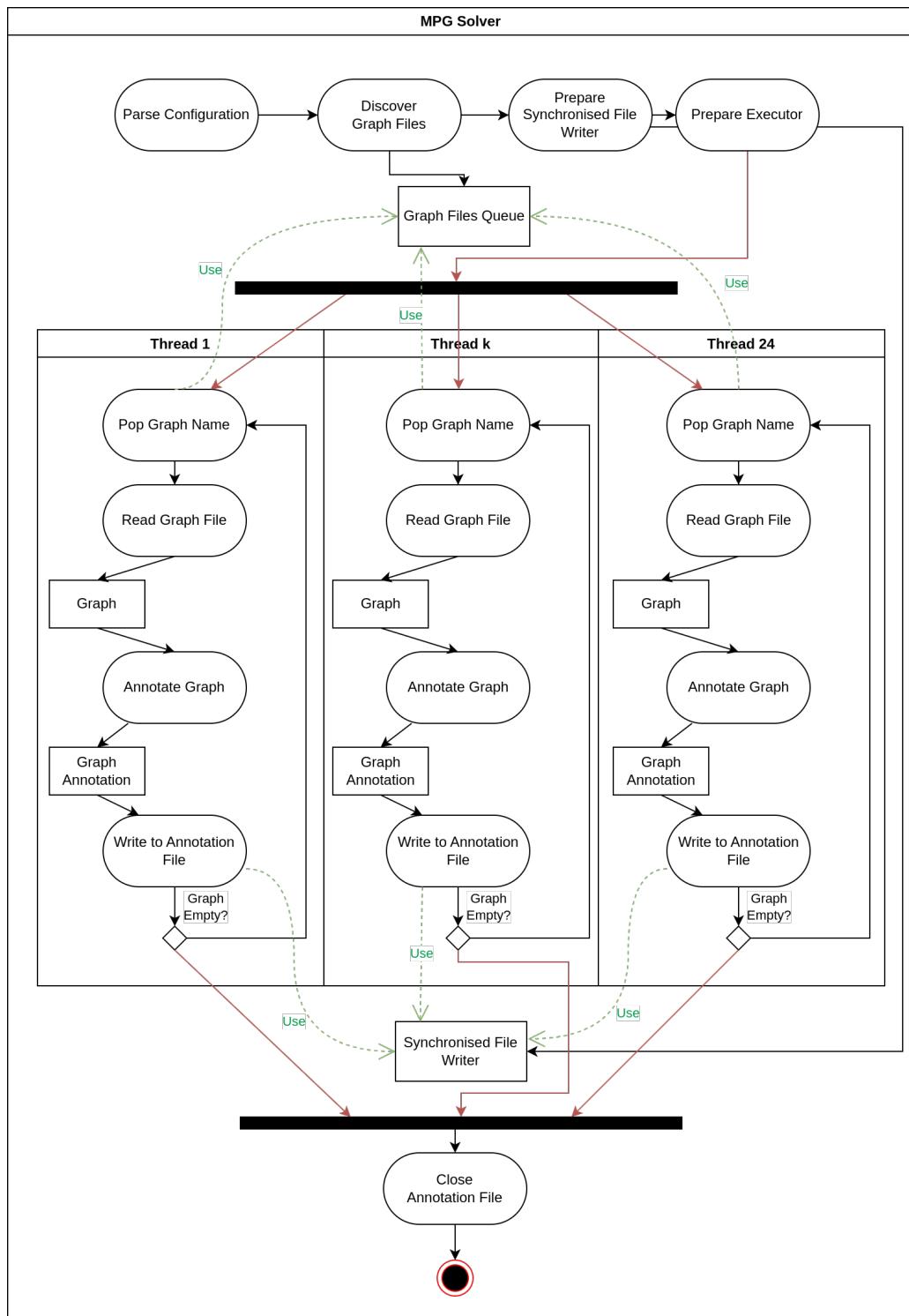


Figure 3.7: Activity diagram describing the behaviour of a node

7.2 Results

Dataset	Size	Cardinality	Annotated	Linear Bound	Early Stopping	Time
Dense	?	160000	160000	✓	✓	
Sparse	?	160000	150000	✓	✗	

Table 3.2: List of heuristics used by the solver

Conclusion

In this chapter, we generated random MPG by decomposing the problem into generating a random graph, and then generating random weights.

We analysed the graph generation problem, and we were able to produce an optimal¹⁵ algorithm for generating a random graph, and then also an optimal graph to generate sinkless graphs.

Then, we proposed desired properties of MPG distributions, and finally implemented some notable one. Once we had all the key elements, we designed and two MPG datasets, one for sparse graphs and one for dense ones. We implemented a graph generator program and deployed it in a HPC system to reduce execution time.

Once the datasets were generated, we were able to also implement a solver by reducing the problem to a CSP. This gave us the possibility to apply powerful CSP methods to find optimal strategies. We then implemented the solver in C++ with emphasis on low overhead.

Finally, we deployed the solver to a HPC system, and created a pipeline to annotate both datasets.

¹⁵In the Big-O sense.

4

Model Design

Introduction

This chapter is dedicated to design and implement a MPG model for predicting good strategies and estimating the winner of the game.

Before achieving that, we will analyse our generated datasets, and empirically verify some assumptions that are primordial to the implementation of the model.

Then, we will formalise some symmetries and some other key properties that our model should verify. We will then implement a Weighted Graph Convolutional Network (WGCN) operator that follow most of the listed properties. Then, we will build the model block by block, and then combine the blocks together to get a full Neural Network (NN) architecture that is based on Graph Neural Network (GNN), and that respect all the listed properties.

Later, we will define our optimization objective, and the optimizer we will use, and then we will list the selected optimizer's hyper-parameters.

Finally, we will put both the model and training hyper-parameters to an external configuration file.

1 Properties

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

1.1 Totality

1.1.1 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..

1.1.2 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 implicitly verified by an encoding of the Mean Payoff Games.

1.2 Node Agnostic

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

1.2.1 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 $\Phi : V_1 \rightarrow V_2$ 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:

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

1.2.2 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.

1.2.3 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 $V' = \{0, \dots, |V| - 1\}$, and E', W', s' defined accordingly.

In fact, the encoding is done implicitly by our model¹.

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

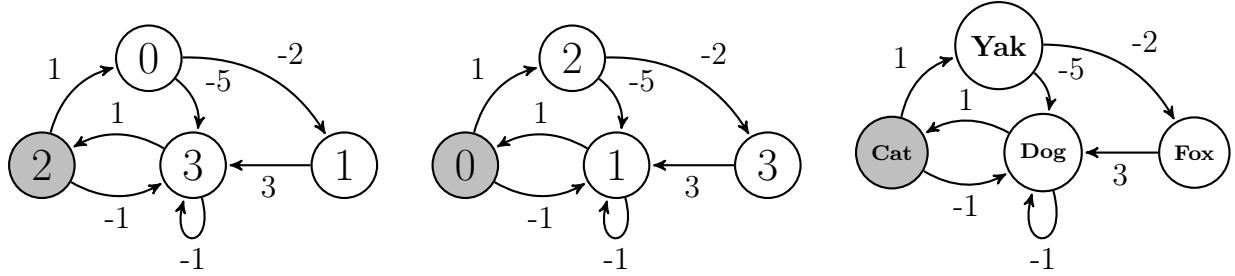


Figure 4.1: Three isomorphic Mean Payoff Games

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:

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

1.3 Invariance under Positive Scaling

This property comes directly from the fact that both the winner and the set of optimal strategies² 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 $H : \mathcal{F}(E, \mathbb{R}) \rightarrow \mathcal{F}(E, \mathbb{R})$ a function satisfying³:

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

²Whatever the definition of optimality (Weak, Strong, Payoff).

³Special care must be when H is \geq instead of $>$

1.3.1 Standard Scaling

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

$$\text{Normalize}(W) = \frac{W}{\sqrt{\mathbb{V}[W]}}$$

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

$$\mathbb{E}[W] = \frac{1}{|E|} \sum_{(u,v) \in E} W(u,v)$$

1.3.2 Maximum Scaling

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

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

1.3.2.1 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 ignore these zeros⁴ in both normalisations, otherwise it may lead to biased scaling.

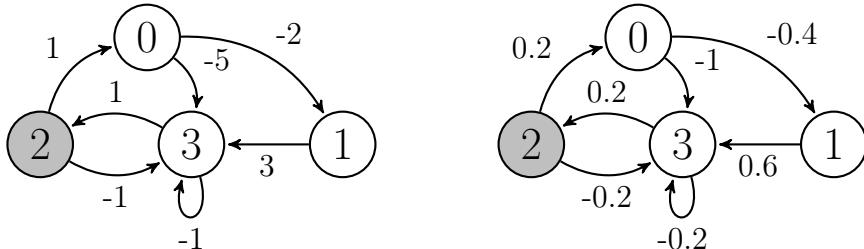


Figure 4.2: Mean Payoff Game, with a rescaled version using Maximum normalization

1.4 Permutation Equivariance

1.4.1 Definition

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

1.4.2 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 implicitly, and thus Permutation Equivariance is enough to get a Node Agnostic model.

⁴ And only these zeros. If $W(u, v) = 0$ for $(u, v) \in E$, this term should be accounted.

1.5 Stability under Padding

1.5.1 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 $V_1 \cap V_2 = \emptyset$.
- 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 $G_1 \xrightarrow{B} G_2$ is defined as:

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

If the bridge B is empty, in the sense that $E_3 = \emptyset$, then we denote we will simplify the notation to $G_1 \triangleright G_2$.

A model is said to be stable under padding if:

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

1.5.2 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 batch⁵, it suffered the following limits:

- 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.

1.5.3 Removing Unreachable Nodes

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

⁵A batch of inhomogeneous input

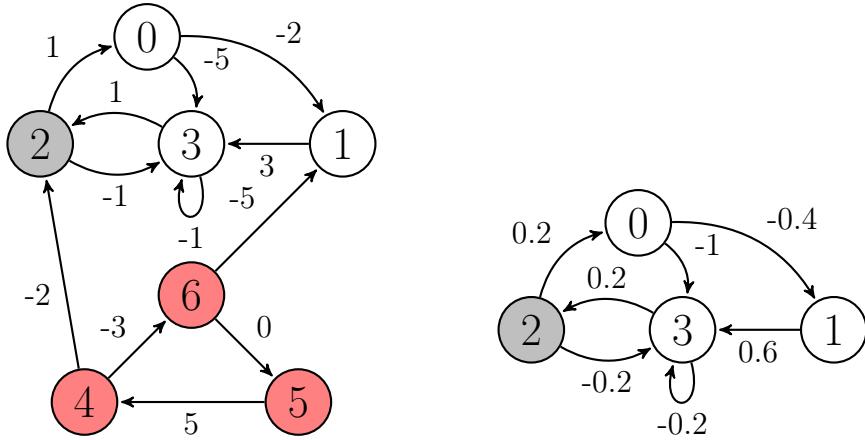


Figure 4.3: Mean Payoff Game before and after removing unreachable nodes

Following the example under figure 4.3, if the model \mathcal{M} is stable under padding, then the vertices $\{4, 5, 6\}$ can be removed without affecting the results.

In fact, this is true as the original mean payoff game can be constructed from the reduced one with a suitable padding.

2 Considered Models

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

2.1 Value based Model

Such model predicts the evaluation of a certain position.

The evaluation is a function $\mathcal{M} : V \times P \rightarrow [-1, 1]$ with the following interpretation:

- $\mathcal{M}(s, p) = 1$ when the model predicts that player p will win the game given the position.
- $\mathcal{M}(s, p) = -1$ when 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 efficient⁶ 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.

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

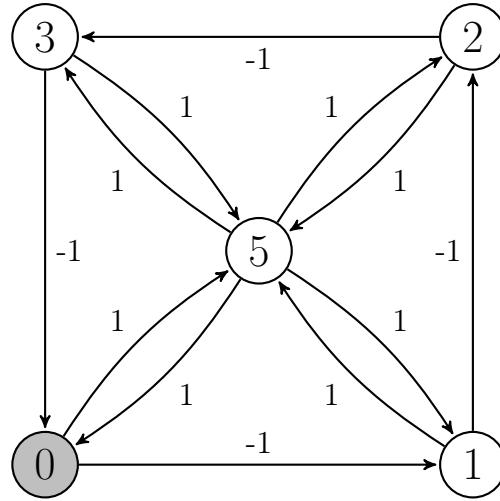


Figure 4.4: An MPG where the first player always wins.

2.2 Strategy based Model

Such model predicts the strategy for each player.

This is a function $\mathcal{M} : V \times P \rightarrow \mathcal{P}(V)$ such that:

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

3 Building the Model

Our model follows from well established graph neural network architectures, with minor modifications to suit our needs.

It is composed of blocks, each containing a **Weighted Graph Convolutional Network**.

3.1 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 homogeneous 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.

3.1.1 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:

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

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

3.1.2 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 $\mathcal{B}(p)$.

3.1.3 Weights Normalisation

We want the model to be invariant under positive scaling. To achieve that, we implemented a weights normalization layer conforming to section 1.5

3.1.4 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 $\mathcal{N}(0, \sigma)$

Now, we should choose the noise small enough so that we do not change the winner of the mean payoff game, nor the optimal strategies.

Theorem 6 For a Mean Payoff Game $G = (V, E, W, s, p)$, with integer weights.

If $\|N\|_\infty < \frac{1}{2|V|}$, then G and $G' = (V, E, W + N, s, p)$ have the same sets of C_3 -optimal strategies for each player.

Preuve 2 Without loss of generality, we assume that Max wins.

Let $\Phi^{\text{Max}} \in \text{Optimal}(G, \text{Max})$ and $\Phi^{\text{Min}} \in \text{Optimal}(G, \text{Min})$

Without a loss of generality, we assume that both $\Phi^{\text{Max}}, \Phi^{\text{Min}}$ are positional.

Let $\mathcal{C} = (u_0, \dots, u_m)$ the cycle of vertices induced by both strategies in G .

As Max wins, we have:

$$v(G, \Phi^{\text{Max}}, \Phi^{\text{Min}}) = \frac{1}{2m} \sum_{k=1}^m W(u_{k-1}, u_k) > 0$$

As all the weights are integers, we have:

$$\sum_{k=1}^m W(u_{k-1}, u_k) \geq 1$$

With that, we conclude the following lower bound of the mean payoff:

$$v(G, \Phi^{\text{Max}}, \Phi^{\text{Min}}) \geq \frac{1}{m} \geq \frac{1}{2|V|}$$

Finally, we show that Max is also winning in G' :

$$\begin{aligned} v(G', \Phi^{\text{Max}}, \Phi^{\text{Min}}) &\geq \frac{1}{2m} \sum_{k=1}^m W(u_{k-1}, u_k) + N(u_{k-1}, u_k) \\ &\geq v(G, \Phi^{\text{Max}}, \Phi^{\text{Min}}) - \|N\|_\infty \\ &> 0 \end{aligned}$$

This result gives the theoretical interval in which the noise will not change the optimal strategies in an integer Mean Payoff Game.

In practice, as N follows a symmetric distribution, the noise will tend to cancel itself, and the mean of the noise will tend to zero.

For that reason, we do believe that a noise $N \sim \mathcal{U}(-0.1, 0.1)$ or $N \sim \mathcal{N}(0, 0.1)$ will keep the winners and strategies intact for most generated mean payoff games⁷.

3.1.5 One Hot Encoding

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

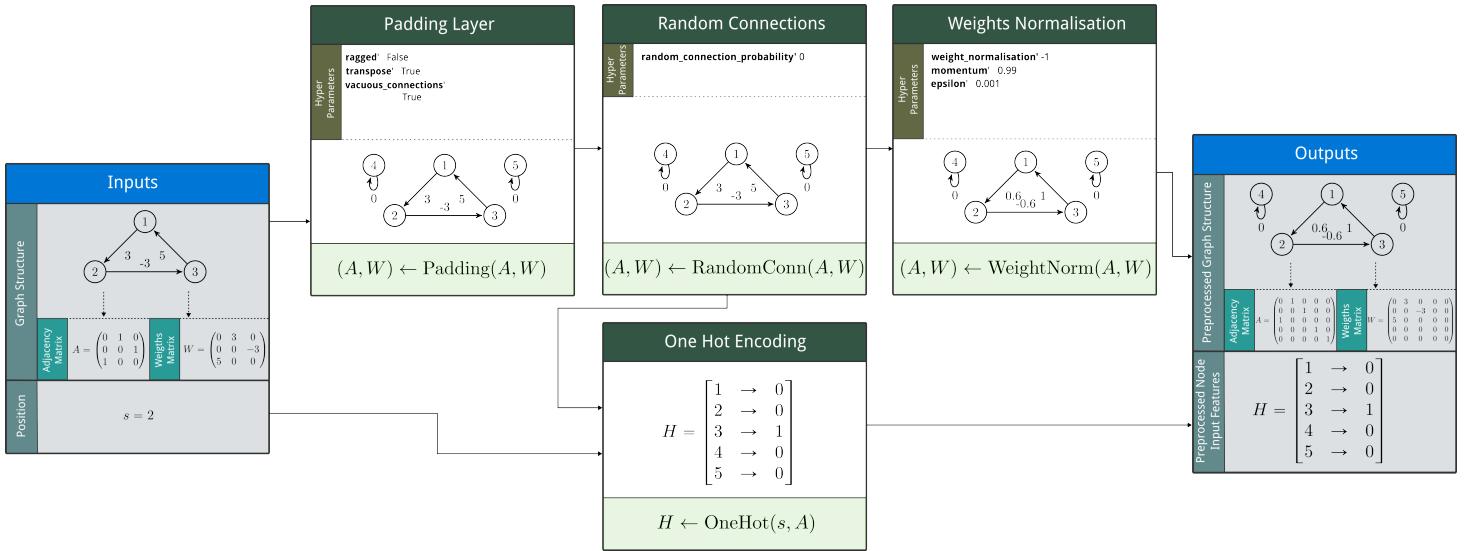


Figure 4.5: Preprocessing Block

⁷Even for mean payoff games with real weights, we do believe that the same applies for weights following the distribution $\mathcal{U}(-1, 1)$.

3.2 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

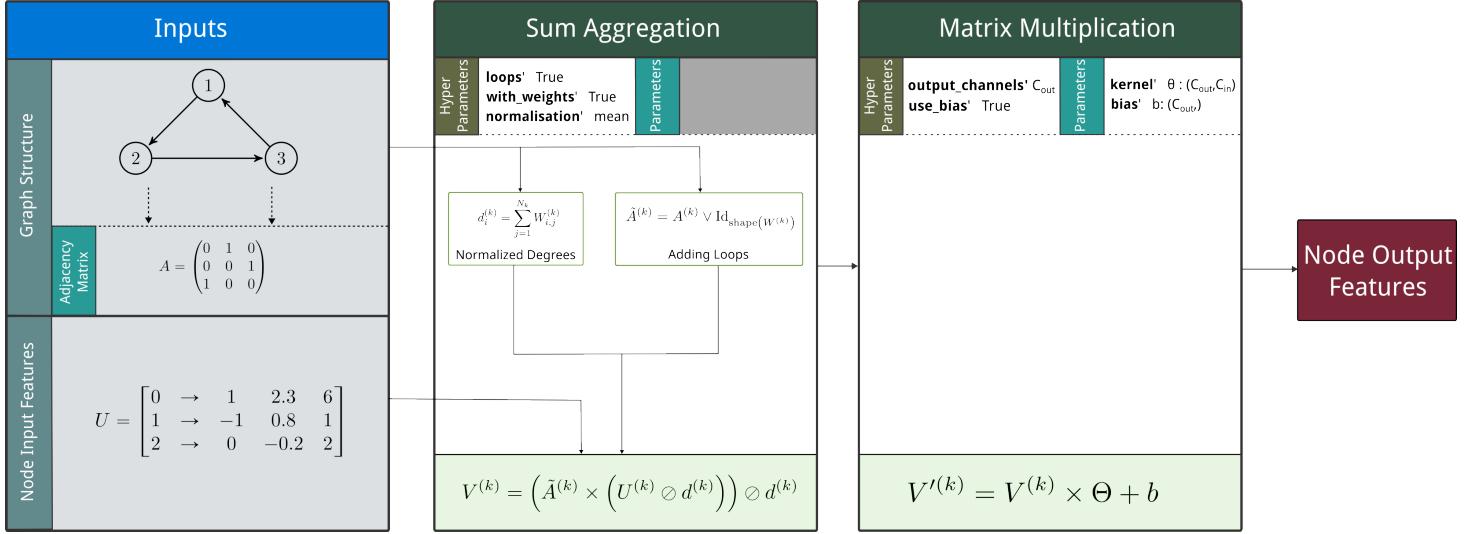


Figure 4.6: Graph Convolutional Network

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.

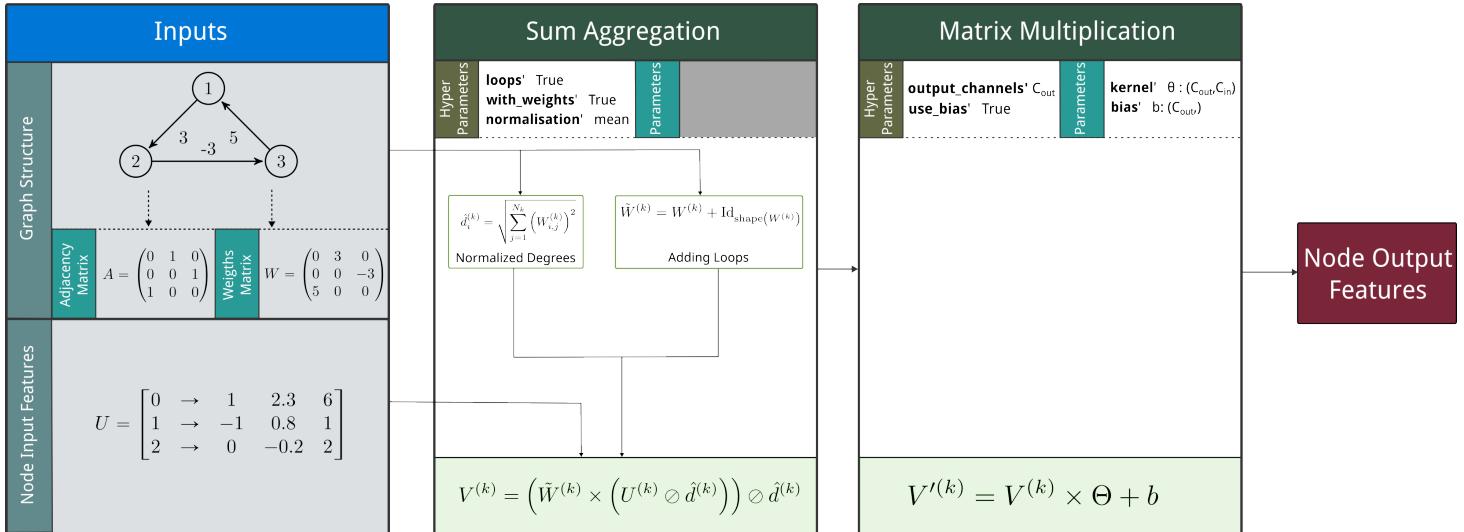


Figure 4.7: Weighted Graph Convolutional Network

The **WGCN** operator exhibits the desired properties described on 1, except property 1.3, 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.

3.3 Convolutional Block

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

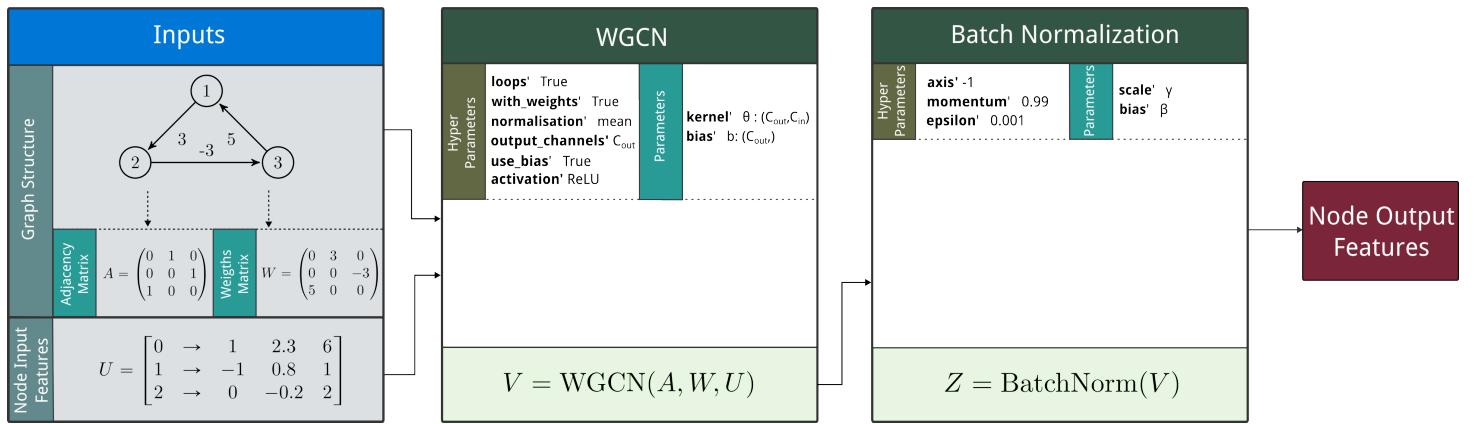


Figure 4.8: Graph Convolutional Block

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

3.4 Prediction Block

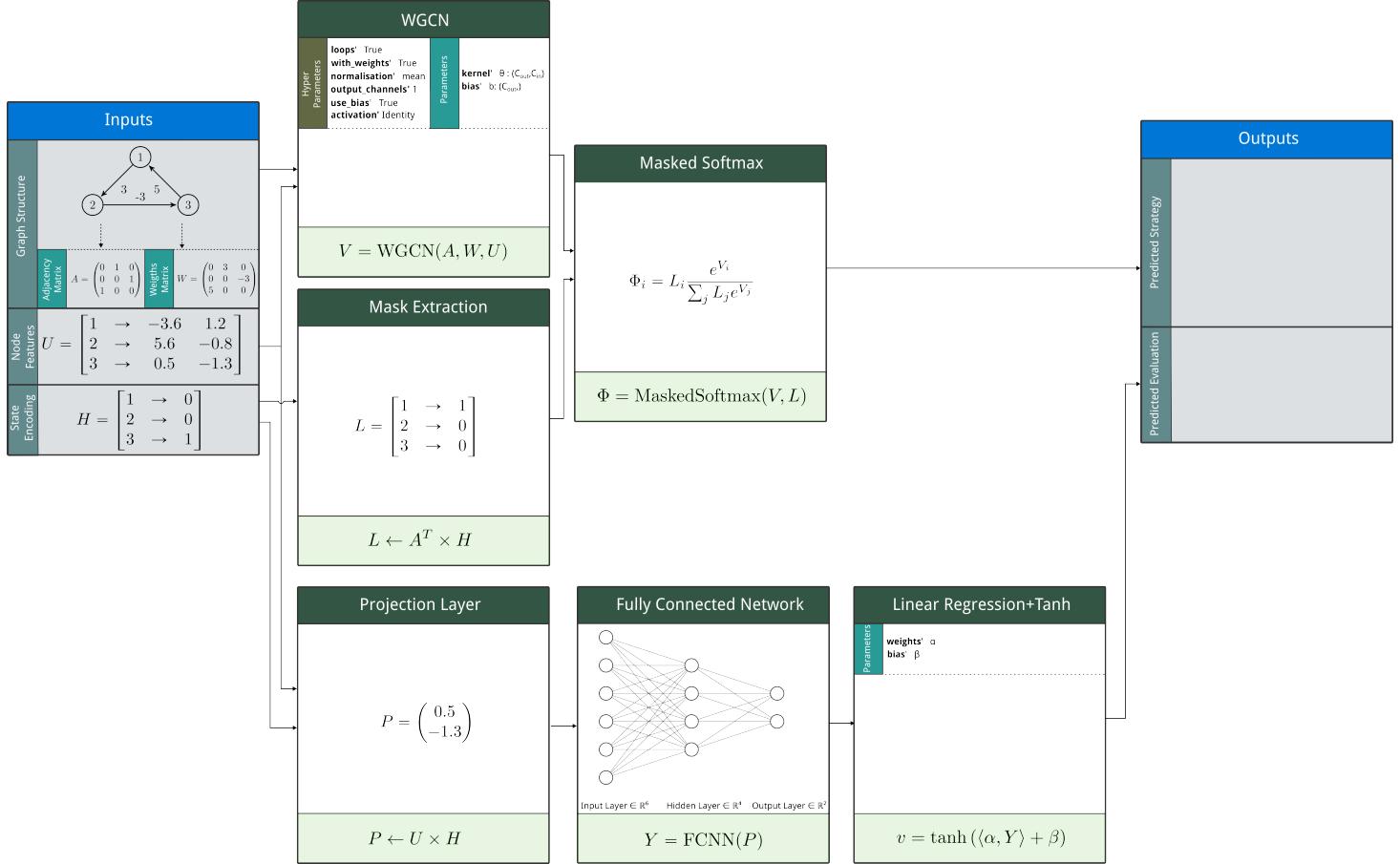


Figure 4.9: Graph Convolutional Block

3.5 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 1.

The figure below shows the whole architecture:

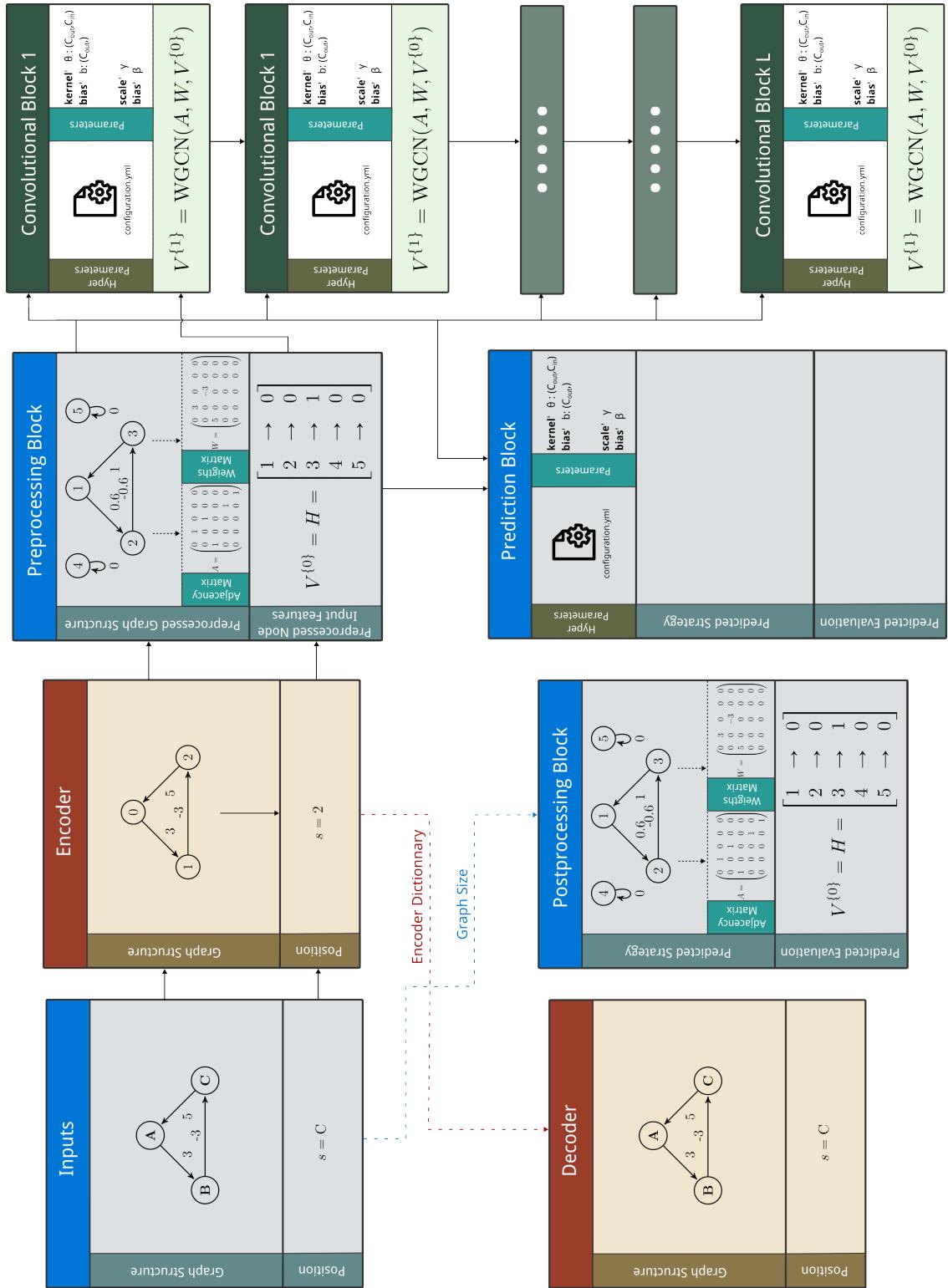


Figure 4.10: Model Architecture

4 Optimization

Once we built the model architecture, we have to define the optimization problem to refine the learnable parameters.

Also, there is a wide range of optimizers that we can use. The next two sections will respectfully define the loss function and the optimizer.

4.1 Loss function

- Let \mathbf{G} be a batch of MPG.
- Let \mathbf{v} be a batch of evaluations of \mathbf{G}
- Let $\mathbf{\Pi}$ be a batch of strategies of \mathbf{G}
- Let \mathcal{M}_Θ be the model with Θ the learnable parameters.
- Let \mathbf{v}_Θ be the predicted evaluation per model \mathcal{M}_Θ
- Let $\mathbf{\Pi}_\Theta$ be the predicted strategy per model \mathcal{M}_Θ
- Let \mathcal{H} be the categorical cross entropy operator defined as follow:

$$\mathcal{H}(p, q) = \sum_i p_i \log q_i$$

We defined the loss function similar to Alpha Zero's approach [45]:

$$\mathcal{L}(\mathcal{M}_\Theta, \mathbf{G}, \mathbf{v}, \mathbf{\Pi}) = \mathcal{H}(\mathbf{\Pi}, \mathbf{\Pi}_\Theta(\mathbf{G})) + \|\mathbf{v}_\Theta(\mathbf{G}) - \mathbf{v}\|_2^2 + C\|\Theta\|_2^2 \quad (4.1)$$

This is the sum of three terms:

- $\mathcal{H}(\mathbf{\Pi}, \mathbf{\Pi}_\Theta(\mathbf{G}))$: This term encodes the error between the predicted strategy $\mathbf{\Pi}_\Theta(\mathbf{G})$ and the actual strategy $\mathbf{\Pi}$.
- $\|\mathbf{v}_\Theta(\mathbf{G}) - \mathbf{v}\|_2^2$: This term encodes the euclidean distance between the predicted evaluation $\mathbf{v}_\Theta(\mathbf{G})$ and the actual evaluation \mathbf{v} .
- $C\|\Theta\|_2^2$ This term adds \mathcal{L}^2 regularization to the model, with C serving as the strength of the regularization.

Also, each error term in the loss function is applied to each game individually. In fact, equation (4.1) is in a compact representation that reduces⁸ to the mean of individual errors in the batch.

4.2 Optimizer

We used the Adam⁹ optimizer [25] to minimize the objective function \mathcal{L} with respect to Θ

⁸Up to some normalization factor, this is an implementation detail.

⁹Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

4.3 Hyper-parameters

Our choice of optimizer and loss function induced more hyper-parameters. We will list them in the following table:

Hyper-parameter	Symbol	Default value	Description
<code>learning_rate</code>	μ	10^{-3}	The step size of the gradient descent method
<code>beta_1</code>	β_1	0.9	The exponential decay rate for the 1 st moment estimates.
<code>beta_2</code>	β_2	0.999	The exponential decay rate for the 2 nd moment estimates.
<code>epsilon</code>	$\hat{\varepsilon}$	10^{-7}	A small additive constant used for numerical stability.
<code>weight_decay</code>	C	10^{-4}	The strength of the \mathcal{L}^2 regularization.

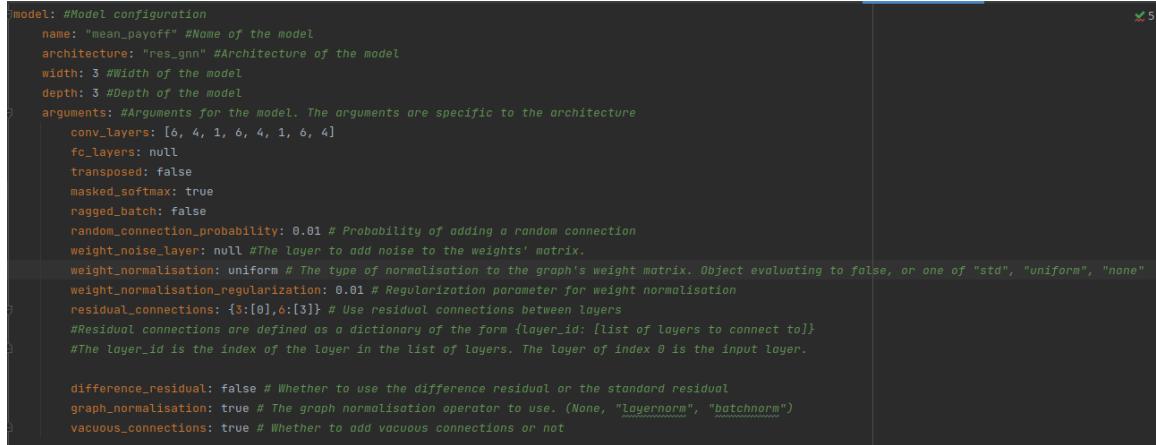
Table 4.1: Le tableau d'avancement des BNNs

5 Configuration

we externalized our configuration so that we can change hyper-parameters and other non-learnable parameters at will, without modifying the source code. This was beneficial as it separated between the model implementation phase and the model fine-tuning phase.

5.1 Model configuration

We setup a YAML configuration file containing the entries for tweaking the model.



```

model: #Model configuration
  name: "mean_payoff" #Name of the model
  architecture: "res_gnn" #Architecture of the model
  width: 3 #Width of the model
  depth: 3 #Depth of the model
  arguments: #Arguments for the model. The arguments are specific to the architecture
    conv_layers: [6, 4, 1, 6, 4, 1, 6, 4]
    fc_layers: null
    transposed: false
    masked_softmax: true
    ragged_batch: false
    random_connection_probability: 0.01 # Probability of adding a random connection
    weight_noise_layer: null #The layer to add noise to the weights' matrix.
    weight_normalisation: uniform # The type of normalisation to the graph's weight matrix. Object evaluating to false, or one of "std", "uniform", "none"
    weight_normalisation_regularization: 0.01 # Regularization parameter for weight normalisation
    residual_connections: {3:[0],6:[3]} # Use residual connections between layers
    #Residual connections are defined as a dictionary of the form {layer_id: [list of layers to connect to]}
    #The layer_id is the index of the layer in the list of layers. The layer of index 0 is the input layer.

    difference_residual: false # Whether to use the difference residual or the standard residual
    graph_normalisation: true # The graph normalisation operator to use. (None, "layernorm", "batchnorm")
    vacuous_connections: true # Whether to add vacuous connections or not

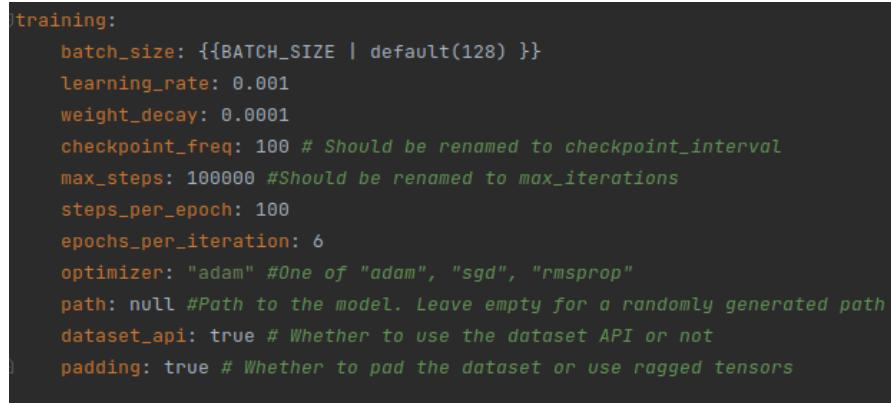
```

Figure 4.11: Model section of the configuration file

Note: while a recent version of the configuration file, we may tweak it further for fine-tuning purposes.

5.2 Training configuration

The configuration file also has a section for training hyper-parameters. We will show it in the next figure:



```

training:
  batch_size: {{BATCH_SIZE | default(128)}}
  learning_rate: 0.001
  weight_decay: 0.0001
  checkpoint_freq: 100 # Should be renamed to checkpoint_interval
  max_steps: 1000000 #Should be renamed to max_iterations
  steps_per_epoch: 100
  epochs_per_iteration: 6
  optimizer: "adam" #One of "adam", "sgd", "rmsprop"
  path: null #Path to the model. Leave empty for a randomly generated path
  dataset_api: true # Whether to use the dataset API or not
  padding: true # Whether to pad the dataset or use ragged tensors

```

Figure 4.12: Training section of the configuration file

Note: while a recent version of the configuration file, we may tweak it further for fine-tuning purposes.

Conclusion

By the end of this chapter, we were able to get a full GNN model for predicting good strategies and evaluating positions.

To do that, we started by analysing both MPG datasets that were generated by the end of 3. This analysis was crucial to verify some of our hypotheses about MPG that will be implied by the model.

Then, we formalised some symmetries and other properties that we wish the model to verify, and implemented the model precisely so that not only the properties will always hold, but also the model will be stable. To achieve that, we implemented a custom WGCN layer that is inspired from the GCN operator. And then we built the model's blocks one by one, and finally joined them together into a full architecture.

Furthermore, we defined our optimization objective in a way consistent with our needs. We also selected our optimizer, and listed its hyper-parameters.

Finally, as the list of hyper-parameters grew larger and larger, we externalized the model and training hyper-parameters to a configuration file.

5

Reinforcement learning and Self play approach

Introduction

In this final chapter, we will define Reinforcement Learning (RL) more formally, and then briefly describe the difference of some RL approaches.

Once we have the RL background, we will transform our MPG into a Stochastic Game (SG) in a consistent way with the model architecture defined on chapter 4.

Then, we will begin on Self Play (SP) system. First, we will describe vanilla MCTS implementations with some required modifications to the SG for convergence, and later, we will discuss the MCTS based on Alpha Zero that uses the model defined on chapter 4.

Later, we will design and implement the SP pipeline, deploy it to a HPC pipeline.

Finally, we will create a configuration file for externalizing the configuration, and then do some experiments with out SP system

1 Reinforcement Learning

1.1 Definition

Sutton and Barto [48, Chapter. 1] gave an excellent in-depth definition, which not only defines what Reinforcement Learning (RL) is, but also compare it with other Machine Learning Learning (ML) methods.

We will summarize it as follow:

Définition 1

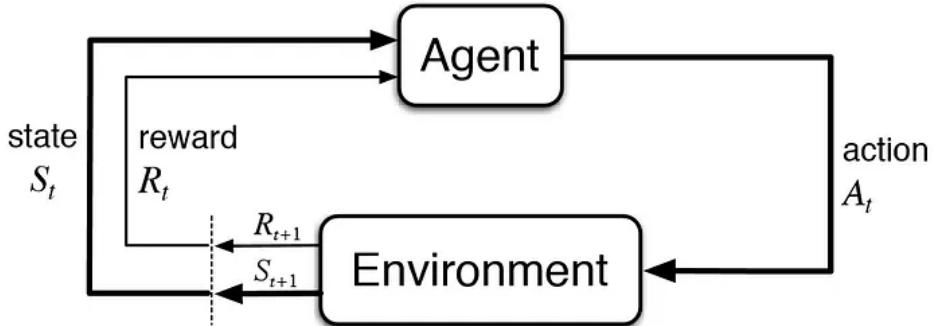


Figure 5.1: A Reinforcement Learning system

1.2 Theory

1.2.1 Single Agent

In most settings, the theory of RL deals with environments where we would like to optimize a single agent. This is called single agent RL. This is usually modeled within a Markov Decision Process (MDP) [48, chapter. 3; 23] AIModernApproach

Now, the sole problem is the agent itself may or maynot know the model¹ of the MDP. Thus there are mainly two approaches:

- **Model-based RL.**
- **Model-free RL.**

Single agent RL applies when we want to find the best counter strategy of player $\text{Opt} \in P$ in a MPG. Model-based RL applies when Opt 's strategy is deterministic² or fractional. If Opt 's strategy is complicated or unknown, we can only use Model-free RL methods.

While our ultimate objective is a model that plays good enough irrespective of the opponent's strategy, our implementation does support for RL-based strategy countering, albeit it does need retraining for every MPG instance.

1.2.2 Multi Agent

When we have multiple agents with potentially conflicting objectives, we call this multi-agent RL.

As the single agent case, each agent in the SG may or may not know the the underlying model³.

In our case, a MPG can be modeled in the RL setting as a turn-based, two-player, zero-sum Stochastic Game (SG)⁴ [46].

We will use the SG formalism for the self-playing part.

¹In this particular context, a model of the MDP is the representation of all transitions, their rewards, and the transition probabilities. That is the complete knowledge of the MDP. This has nothing to do with our meaning of model.

²For the deterministic case, RL is an overkill. We can fallback to negative cycle finding.

³Model as in the MDP sense.

⁴Also known as Markov Game

1.3 RL formalisation of a MPG

To formalise a MPG in RL setting, we have two formalisms. Before diving into that, we will use the MPG definition in section 1.2:

$$G = (V, E, W, s, p)$$

1.3.1 Instance based formalism

In this formalism, we the SG is directly defined by the MPG.

That is, th

1.3.2 Global formalism

In this formalism, we will augment the SG to account to the space of all MPG.

This will require the following definition:

Définition 2 *The set of all MPG, denoted by \mathbb{M} is the set generated by $G = (V, E, W, P, s, p)$ where:*

1. $\emptyset \subsetneq V \subset_{finite} \mathbb{N}$
2. $E \subseteq V \times V$ with the sinkless requirement:

$$\forall u \in V, \quad \text{Adj } u \neq \emptyset$$

3. $W \in \mathcal{F}(E, [-1, 1])$

4. $s \in V$.

5. $p \in P$

This set contains all MPG up to isomorphism and positive rescaling.

In this sense, we can play G in \mathbb{M} as follow:

- We set $G \in \mathbb{M}$ as the starting state.
- Let $G = (V, E, W, P, s, \text{Opt})$ be the current node, then $\text{Opt} \in P$ is the current player, his available actions are $\text{Adj } s$
- Suppose that Opt chose action $t \in \text{Adj } s$, then he gets a payoff of $W(s, t)$, and the state transitions to $G' = (V, E, W, P, t, \bar{\text{Opt}})$

In fact, the game that we have just proposed is the same as the following MPG $\mathcal{G} = (\mathbb{M}, \mathbb{E}, \mathbb{W}, V, G, p)$ with:

$$\mathbb{E} = \{(V, E, W, P, s, \text{Opt}) \rightarrow (V, E, W, P, t, \bar{\text{Opt}}) / (s, t) \in E\}$$

$$\forall G \rightarrow G', \quad \mathbb{W}(G, G') = W(s, t) \quad \text{Where } \begin{cases} s & \text{is the starting vertex of } G \\ t & \text{is the starting vertex of } G' \end{cases}$$

Now, it is trivial to see that the MPG \mathcal{G} is equivalent to G , and here we put all this

theory into effect: **The RL methods will be done in \mathcal{G} instead of G .**

1.4 Benefits of global formalism

At first glance, it seems that we have only complicated the matter. In fact, this is justified as we can only effectively apply function approximation methods RL using the global formalism.

To clarify our point, we recall that in chapter 4, we designed a model \mathcal{M}_Θ that takes a MPG G and outputs a strategy Π_Θ and an evaluation v_Θ . We will use RL to refine Θ so that:

- $v_\Theta(G)$ approaches the value of the game⁵ $v(G)$.
- $\Pi_\Theta(G)$ approaches an optimal strategy Π of game G .

Now, with the right choice of RL algorithm, we expect that for any close enough⁶ game G' :

- $v_\Theta(G')$ will also approach the value of the game $v(G')$.
- $\Pi_\Theta(G')$ will also approach an optimal strategy Π of game G' .

This is crucial: **We are training \mathcal{M}_Θ using RL so that it generalises to any MPG.**

In contrast, if we just use the instance based formalism, the underlying model can only take the current position as information, and thus it does only generalises to solving the same game, but with only varying starting positions.

1.5 Generating random MPG

This is the last thing to be addressed before implementing the self-playing pipeline.

To expect our model to generalises well to any MPG. We should generate the games using a suitable distribution. To do that, we introduced two additional distributions:

1.5.1 $\mathcal{D}_{p,\text{US}}^\bullet$ distribution

- Let \mathcal{N} be a non-empty finite subset of \mathbb{N} .
- Let \mathcal{P} be a measurable subset of $[0, 1]$.
- For a measurable set S , we will denote by $\mathcal{U}(S)$ the uniform distribution on S .

The $\mathcal{D}_{p,\text{US}}^\bullet(\mathcal{N}, \mathcal{P})$ is a mixture distribution defined as follow:

$$\mathcal{D}_{p,\text{US}}^\bullet(\mathcal{N}, \mathcal{P}) = \mathcal{D}^\bullet(N, P) \quad \text{where} \quad \begin{cases} N & \sim \mathcal{U}(\mathcal{N}) \\ P & \sim \mathcal{U}(\mathcal{P}) \end{cases}$$

To sample an element from $\mathcal{D}_{p,\text{US}}^\bullet(\mathcal{N}, \mathcal{P})$:

⁵The definition is stated in section

⁶To formalise this idea, we need to define a metric space in \mathcal{G} . This is beyond the scope of this report, but the point still holds.

1. Select the graph size $N \in \mathcal{N}$ uniformly at random.
2. Select the edge probability $p \in \mathcal{P}$ uniformly at random.
3. Build $G \sim \mathcal{D}^*(N, P)$ using algorithm 4 and degree rejection method defined at section 3.2.

1.5.2 $\mathcal{D}_{c,US}^*$ distribution

- Let \mathcal{N} be a non-empty finite subset of \mathbb{N} .
- Let \mathcal{C} be a non-empty finite subset of \mathbb{N} .

The $\mathcal{D}_{c,US}^*(\mathcal{N}, \mathcal{C})$ is a mixture distribution defined as follow:

$$\mathcal{D}_{c,US}^*(\mathcal{N}, \mathcal{C}) = \mathcal{D}^*(N, \frac{C}{N}) \quad \text{where} \quad \begin{cases} N & \sim \mathcal{U}(\mathcal{N}) \\ C & \sim \mathcal{U}(\mathcal{C}) \end{cases}$$

To sample an element from $\mathcal{D}_{c,US}^*(\mathcal{N}, \mathcal{C})$:

1. Select the graph size $N \in \mathcal{N}$ uniformly at random.
2. Select the expected degree $c \in \mathcal{C}$ uniformly at random.
3. Build $G \sim \mathcal{D}^*(N, \frac{C}{N})$ using algorithm 4 and degree rejection method defined at section 3.2.

1.5.3 Importance

In chapter 3, we designed and implemented a graph generation pipeline, then a graph annotation pipeline for dense graphs and later sparse graphs.

Both datasets $\mathcal{D}^{\text{dense}}$ and $\mathcal{D}^{\text{sparse}}$ contain the evaluation of each game and its respective optimal strategy. We can use both datasets for evaluation purposes, and for that we will make sure that our learns from the same distribution.

Following our implementation in chapter 3:

- Each generated dense graph follow $\mathcal{D}_{p,US}^*(\mathcal{N}, \mathcal{P})$. with \mathcal{N} and \mathcal{P} defined on section 5.1, at the **Dense Graphs** paragraph.
- Each generated sparse graph follow $\mathcal{D}_{c,US}^*(\mathcal{N}, \mathcal{C})$ with \mathcal{N} and \mathcal{C} defined on section 5.1, at the **Sparse Graphs** paragraph.

1.6 Considered RL algorithm

Now, all the technicalities about the formalisation of MPG in RL setting are resolved. Also, we have a complete model \mathcal{M} as a result of chapter 4.

The remaining part is the choice of the RL itself. As hinted by previous sections, we will implement a self-playing system based on Alpha Zero [45].

This system will indefinitely:

- Generate random instances of a MPG, and annotate them with the model \mathcal{M}_Θ using self-play.
- Sample a dataset \mathcal{D} from the generated annotated games, and fit the model \mathcal{M}_Θ to get a refined model $\mathcal{M}_{\Theta'}$.
- Evaluate the model against a fixed set of opponents.

With all that said, only the self-playing system is remaining. Thus in the upcoming sections, we will:

1. Formalise the self-playing approach.
2. Fully implement it.
3. Deploy it in a HPC cluster.
4. Execute the whole pipeline and fine-tune the model.

2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an advanced algorithm in decision making that appeared in 2007. Coulom first coined the term MCTS as application of Monte-Carlo methods to game-tree search [11]. It is an important algorithm for adversarial search. that had a huge success at improving the competence of Go engines to a master level [15]. This alone was a feat considering that previous state of the art alpha-beta implementations were unable to get past beginner level.

Much development occurred to MCTS, which made it a little hard to define. Sutton and Barto gave the following definition [48, section. 8.8]: “*Monte Carlo Tree Search (MCTS) is a planning algorithm that accumulates value estimates obtained from Monte Carlo simulations in order to successively direct simulations towards more highly-rewarded trajectories.*”

This definition alone is unsatisfactory, Russell gave an intuitive explanation [40, Section 6.4] of how Monte Carlo Tree Search (MCTS) works for a game. And we will base this section on his work.

Essentially, MCTS consists of the following 4 steps:

2.1 Selection

Starting at the root, we choose a child following a **selection policy** $\Pi^{\text{selection}}$, and repeat the step until arriving to a leaf node.

Algorithm 10 Selection algorithm for MCTS

Require: r the root of the MCTS.
Require: $\Pi^{\text{selection}}$ the current selection policy..
Ensure: s a leaf of the MCTS.

```

 $s \leftarrow r$ 
while not leaf( $r$ ) do
     $s \leftarrow \Pi^{\text{selection}}(r)$ 
end while
return  $s$ 

```

2.2 Expansion

At the selected leaf s , we grow the MCTS by adding a new child t that is a result of some action a from s .

2.3 Simulation

Starting from the new child t , we perform a complete playout, that is, we choose moves for all players according to the playout policy Π^{playout} .

The generated sequence of nodes in the playout will not be recorded in the MCTS

Algorithm 11 Expansion algorithm for MCTS

Require: s a leaf MCTS.

Require: H the mean reward of each node in the MCTS

Require: T the number of simulations for each node in the MCTS

Ensure: t a new child of s

```

 $C \leftarrow \text{expand\_children}(s)$                                  $\triangleright$  Generate a set of children from  $s$ 
 $\text{for } c \in C \text{ do}$ 
     $T(c) \leftarrow 0$                                                $\triangleright$   $c$  is currently visited 0 times
     $H(c) \leftarrow \mathbf{0}$                                           $\triangleright$  The reward of each player at  $c$  is initialized to 0
 $\text{end for}$ 
 $t \leftarrow \text{choose}(C)$                                           $\triangleright$  Choose a child from  $C$ 
 $\text{return } t$ 

```

Algorithm 12 Simulation algorithm for MCTS

Require: t the new child of the MCTS.

Require: Π^{playout} the current playout policy..

Ensure: W the cumulative rewards after the playout for each player.

```

 $s \leftarrow \text{state}(t)$                                           $\triangleright$  get the current state
 $p \leftarrow \text{state}(t)$                                           $\triangleright$  get the current player
 $W \leftarrow \mathbf{0}$ 
 $\text{while not } \text{terminal}(s) \text{ do}$ 
     $a \leftarrow \Pi^{\text{playout}}(s, p)$ 
     $(s, p, U) \leftarrow \text{apply}(s, a)$                                  $\triangleright$  Apply action  $a$  at state  $s$ , and getting reward  $U$ 
     $W \leftarrow W + U$ 
 $\text{end while}$ 
 $\text{return } W$ 

```

2.4 Backpropagation

Once a simulation is complete, we use its result to update the all search nodes going from t up to the root.

Algorithm 13 Backpropagation algorithm for MCTS

Require: t the new child of the MCTS.
Require: W the reward after the simulation.
Require: W' the cumulative reward up to node t
Require: H the mean reward of each node in the MCTS
Require: T the number of simulations for each node in the MCTS

```

while not root( $s$ ) do
     $T(s) \leftarrow T(s) + 1$                                  $\triangleright$  Increment the number of visits of  $s$ 
     $H(s) \leftarrow H(s) - \frac{W+W'-H(s)}{T(s)}$            $\triangleright$  Update the mean reward at  $s$ 
     $s \leftarrow \text{parent}(s)$                              $\triangleright$  Go to the parent node
end while
return  $W$ 

```

2.5 Wrapping up

Once we have the 4 elements of MCTS, the full algorithm will be just repeating these steps in order until certain desired condition is satisfied. In practice, most implementations use the following conditions for halting the algorithm:

- Execution time exceeds some threshold T
- Total number of simulations (at root) exceeds some threshold N

Algorithm 14 MCTS Algorithm

Require: state some state of the game
Ensure: An action $a \in \text{actions}(\text{state})$

```

 $r \leftarrow \text{tree}(\text{state})$ 
while Exit condition not satisfied do
     $s \leftarrow \text{select}(r)$ 
     $t \leftarrow \text{expand}(s)$ 
     $W \leftarrow \text{simulate}(t)$ 
    backpropagation( $t, W$ )
end while
return  $a \leftarrow \underset{a \in \text{actions}(\text{state})}{\text{argmax}} T(a)$                                  $\triangleright$  Get the most visited action

```

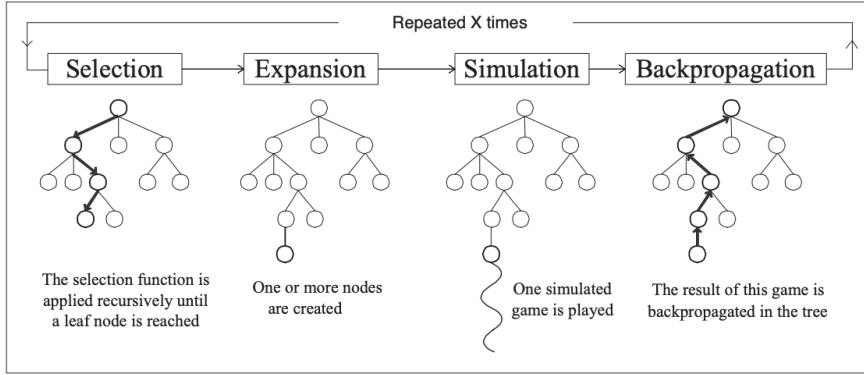


Figure 1: Outline of a Monte-Carlo Tree Search.

Figure 5.2: One iteration of a MCTS algorithm

3 Model based MCTS

While we explained how MCTS works in the previous section, some components were left without much discussion:

1. The selection policy $\Pi^{\text{selection}}$
2. The playout policy Π^{playout}
3. Children creation

This was deliberate, as the full power of *MCTS* requires a good choice of the components, especially $\Pi^{\text{selection}}$ and Π^{playout} .

In our case, both functions will be based on the model \mathcal{M} that we designed in chapter 4 and illustrated in figure 4.10.

Remember that our model \mathcal{M} takes a MPG⁷, and produces two outputs:

1. The predicted evaluation v .
2. The predicted strategy Π .

Both of these terms will be used in the MCTS.

3.1 Create children

In this version of MCTS, create children will return the list of all adjacent states. It does also apply an optional Dirichlet noise $\mathcal{D}(\alpha)$ if the current node is root.

⁷In fact, it takes a batch of MPG, but this can be easily mitigated.

Algorithm 15 Create children

Require: h current node
Require: Dirichlet noise parameter $\alpha \in \mathbb{R}_+$
Require: Exploration noise parameter $\varepsilon \in [0, 1]$
Require: Prior distribution Π
Ensure: A list of children C
Ensure: Optionally update Π

$G = (V, E, W, s, p)$ is the underlying MPG related to node h
 $m \leftarrow |\text{Adj } s|$
if $\text{root}(h)$ **then**
 generate $\nu \sim \mathcal{D}(\alpha \mathbf{1}_m)$ $\triangleright \mathbf{1}_m \in \mathbb{R}^m$ is the vector of ones
 for $(v, x) \in \text{zip}(\text{Adj } s, \nu)$ **do**
 $\Pi(v) \leftarrow (1 - \varepsilon)\Pi(v) + \varepsilon x$
 end for
end if
 $C \leftarrow \emptyset$
for $t \in \text{Adj } s$ **do**
 $G' \leftarrow (V, E, W, t, \bar{p})$
 append(C, G')
end for
return C

3.2 Playout Policy

In Alpha Zero's implementation of MCTS [44], the playout policy is exactly the predicted distribution Π .

Also, the simulation algorithm 12 was modified to only play one move:

Algorithm 16 Alpha Zero MCTS Simulation

Require: t the new child of the MCTS.
Require: \mathcal{M} the neural network model
Ensure: W the cumulative rewards after the playout for each player.

$G \leftarrow \text{MPG}(t)$ \triangleright get the current state
 $p \leftarrow \text{player}(t)$ \triangleright get the current player
if p is Min **then**
 $G \leftarrow \bar{G}$
end if
 $(v, \Pi) \leftarrow \mathcal{M}(G)$
if p is Min **then**
 $v \leftarrow -v$
end if
 $W \leftarrow v$ **return** W

3.3 Selection Policy

Algorithm 17 model-based MCTS playout policy

Require: h current node
Require: $C \in \mathbb{R}_+$ exploration parameter
Ensure: An child MPG G'

```

 $Z \leftarrow 0$ 
for  $c \in \text{children}(h)$  do
     $Z(c) \leftarrow \text{PUCT}(c, h, \Pi(c), C)$ 
end for
return  $c \leftarrow \underset{c \in \text{children}(h)}{\text{argmax}} Z(c)$ 

```

Here PUCT is defined as follow:

$$\text{PUCT}(c, h, \Pi(c), C) = H(c) + C \cdot \Pi(c) \cdot \frac{\sqrt{N(h)}}{N(c) + 1} \quad (5.1)$$

Here $H(c)$ is the reward of the current player at node c .

The equation (5.1) was defined in the Alpha Go paper [44] as a model-based variant of UCT [27]. We show the latter's expression to show the differnece between vanilla MCTS and model-based MCTS:

$$\text{UCT}(c, h, C) = H(c) + C \cdot \frac{\log N(h)}{N(c) + 1}$$

In fact, the additional term $\Pi(c)$ is used to weight the children selection by the prior distribution Π , and we believe that using the square root instead of the logarithmic function was deliberate to boost exploration.

3.4 Wrapping up

Our model-based MCTS is heavily inspired from the one used at Alpha Zero.

It uses the algorithm 14 with a threshold number of iterations N as exit condition⁸. It avoids complete rollouts, and instead uses the simulation algorithm 16 that uses the model \mathcal{M} .

Also, the predictions of model \mathcal{M} are directly used in the selection policy defined at section 3.3 and the playout policy defined at section 3.2.

3.5 Updating model

Once the MCTS algorithm terminates. We get for each node a new estimates of both the strategy and the evaluation.

⁸This applies to the model training phase. While deploying the algorithm, we can use a different exit condition.

3.5.1 Evaluation estimate

The term H used in the MCTS is used for the expected reward for the algorithm. Now as each node t of the tree encodes a MPG G , $H(t)$ will be the updated estimate of the evaluation:

$$v(G) = \mathcal{M}(G)_{\text{evaluation}}$$

For that, we will denote the updated value as $v'(G) = H(t)$

3.5.2 Strategy estimate

To illustrate our point, we will need the following definitions:

- Let t be a node.
- Let $G = (V, E, W, s, p)$ be the MPG encoded by t .
- Let $a \in \text{Adj } s$ be an action eligible from t .
- Let c be the⁹ node resulting from applying a

Now, we want an esimtate of the strategy implied by the MCTS. In fact, the term N used for total explorations helps us to achieve this as following:

$$\Pi'(G)(a) = \frac{N(c)}{N(t)} \quad (5.2)$$

Here, $\Pi'(G)$ will be the refined estimate of $\Pi(G)$

3.5.3 Refitting

- Let Θ be the learnable parameters of $\mathcal{M}_\Theta = (v_\Theta, \Pi_\Theta)$
- Let $G^{(1)}, \dots, G^{(m)}$ a batch of MPG.

We will execute the MCTS on each game $G^{(i)}$ with the model \mathcal{M}_Θ to get a finer estimate of the evaluation $v'_\Theta(G^{(i)})$ and the strategy $\Pi'_\Theta(G^{(i)})$.

Once, we generate enough samples with MCTS. We train the model \mathcal{M}_Θ against \mathcal{D} defined as follow:

$$\mathcal{D} = [(G^{(1)}, v'_\Theta(G^{(1)}), \Pi'_\Theta(G^{(1)})), \dots, (G^{(m)}, v'_\Theta(G^{(m)}), \Pi'_\Theta(G^{(m)}))]$$

The training uses the objective function defined on.

This will give us new learnable parameters Θ' , and thus a new model $\mathcal{M}_{\Theta'}$. With that, we will repeat the process indefinitely.

⁹It is unique as an MPG is deterministic. For a stochastic MPG, which is out of the scope of this report, one should do a more careful analysis.

4 Services

4.1 Rationale

In section 3, we discussed our model-based implementation of MCTS. Roughly speaking, it can be divided into two main components that are repeated indefinitely:

1. Executing MCTS using model $\mathcal{M}_{\Theta^{(i)}}$ to generate a dataset \mathcal{D} :

$$\mathcal{D} = [(G^{(1)}, v'_\Theta(G^{(1)}), \Pi'_\Theta(G^{(1)})), \dots, (G^{(m)}, v'_\Theta(G^{(m)}), \Pi'_\Theta(G^{(m)}))]$$

2. Fit $\mathcal{M}_{\Theta^{(i)}}$ against \mathcal{D} to get a better model $\mathcal{M}_{\Theta^{(i+1)}}$

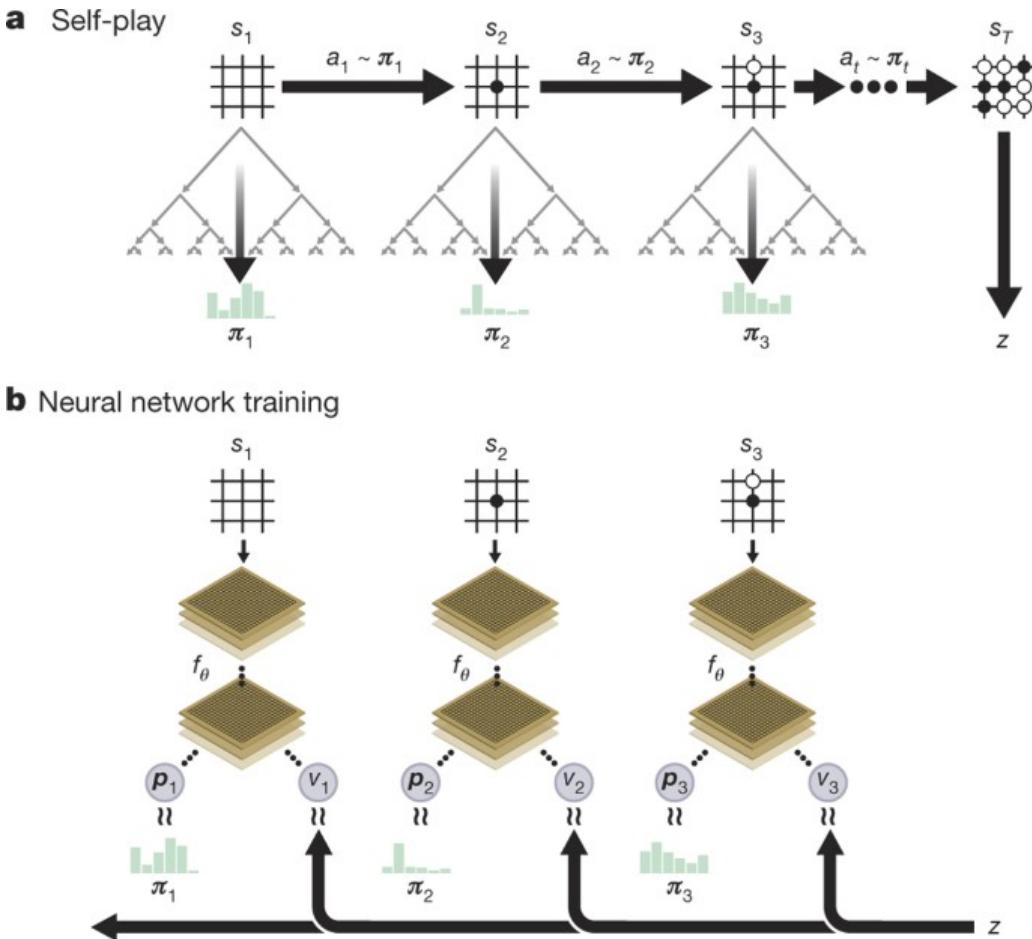


Figure 5.3: Main steps of Alpha Zero

Each one of these steps is computationally intensive. And for that, we will use the separation of concerns principle to split them into distinct services.¹⁰

4.2 Learner

The learner, as its name suggests, is the service that trains the model \mathcal{M} upon receiving enough samples from the actors¹¹

¹⁰This separation, while not necessary, is beneficial as it gives the freedom to scale the self-play pipeline.

¹¹Which we will define in the next section.

4.3 Actor

An actor, is a service designed for generating samples to the learner by playing against itself using model-based MCTS

In contrast to the learner, which is unique by design. The self-play pipeline can have many actors, and this is required in our case as it accelerates the execution of the whole pipeline.

4.4 Evaluator

This service, is a third component that is used to evaluate the performance of model-based MCTS against a fixed player.

Like the actor, a self-play pipeline can have many evaluators, and this is also required in our case as it gives performance metrics against a wide range of players.

Now, we defined the services and talked briefly about their relations. The figure below will describe the communication between the services:

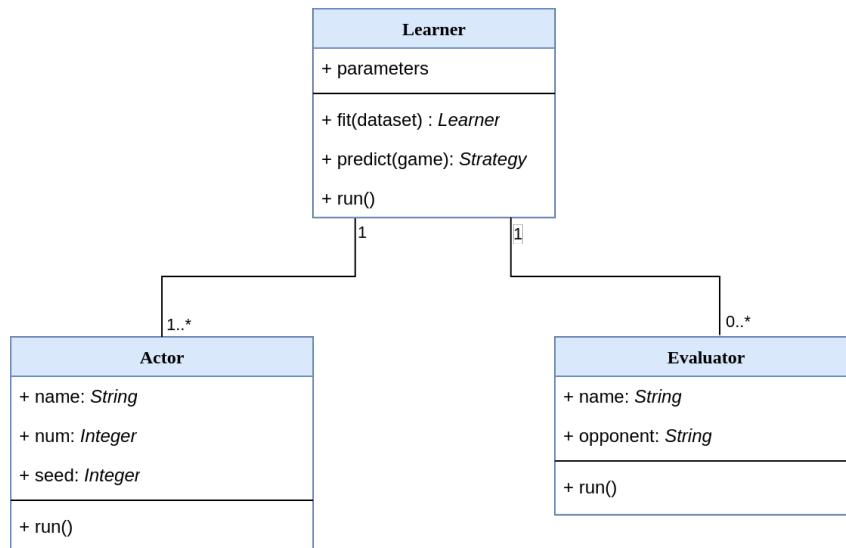


Figure 5.4: Relation between different services

5 Self-play pipeline

5.1 Decoupling service communication

In the previous section, we separated logically between the different components of the self-play system.

Now, this separation will give us the benefit of having multiple actors and evaluators per instance. While this is beneficial, it is only vertically scalable.

In fact, the interactions in figure 5.4 hints that the relation between components is tightly coupled. This is not the case, as we applied the **broker** design pattern to further decouple the communication between the different components of the system.

To do that, we introduced:

- The **Orchestrator**: Which accumulate the generated annotated games and send them to the learner. It is a part of the actor service.
- The **ReplayBuffer**: Which loads a dataset from the annotated games that were generated by actors. It is part of the learner service.
- The **ModelBroadcaster**: It broadcast the new model to all actors and evaluators. It is part of the learner service.

The figure below details the relation between all main components of the pipeline:

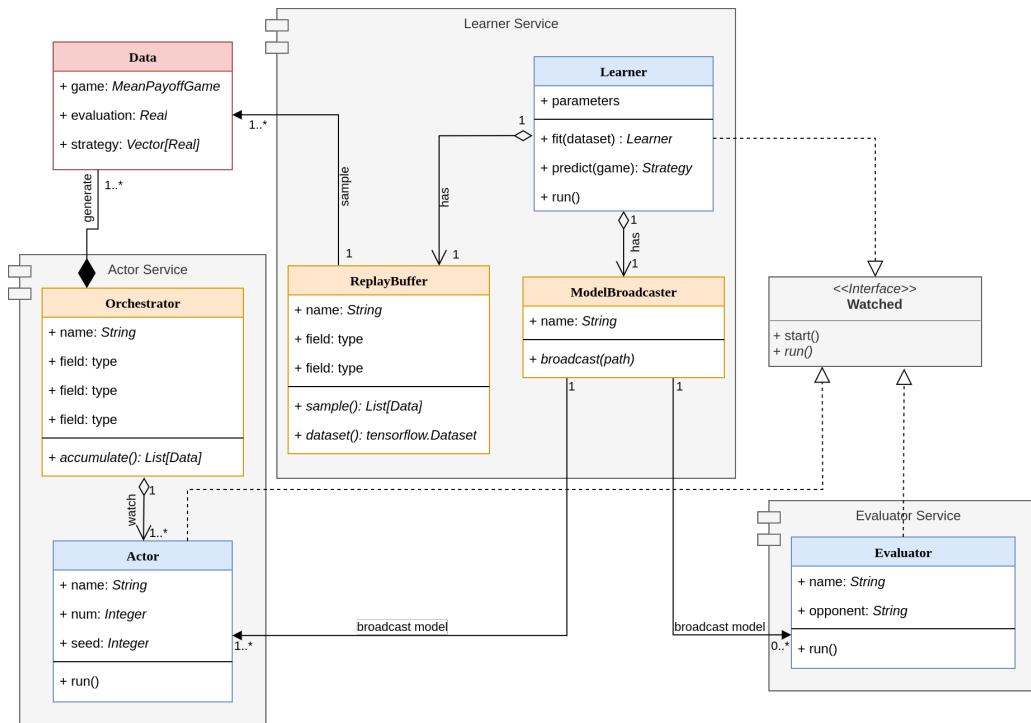


Figure 5.5: Class diagram of the services

5.2 Distributing the pipeline

As we decoupled the communication, we only have to implement concrete definitions of the **Orchestrator**, **ReplayBuffer** and **ModelBroadcaster**.

Currently, we do have two set of implementations:

- The first one defines the communication between the different services when they are centralized¹².
- The second one defines the communication between the different services when they are distributed

The first implementation is present for compatibility reasons with the original implementation offered by `open_spiel`.

¹²Executed in the same machine.

We distributed our pipeline by using the Http protocol¹³. In fact, we used both REST and gRPC. Additionnally, both of **Orchestrator** and **ReplayBuffer** use gRPC, while **ModelBroadcaster** uses REST. This will give us the opportunity to add horizontal scaling for the pipeline, by adding further actor nodes¹⁴ and evaluator nodes.

The next two sections are dedicated respectfully to the REST server and the gRPC server.

5.3 REST servers

Each service¹⁵ deploys a basic **FastAPI** Representational State Tranfer (REST) server used mainly for:

- Basic management: starting the pipeline or shutting the pipeline.
- Service discovery.
- Model broadcasting.
- Monitoring.

The learner actor serves as the master of the services. It will give instructions to the other services. It also serves as the gateway.

Route	Method	Service	Description
/start	GET	All	If the service is the learner, start the whole pipeline by requesting /start to all other services. Otherwise, start the requested service.
/stop	GET	All	If the service is the learner, stop the whole pipeline by requesting /stop to all other services. Otherwise, stop the requested service.
/heartbeat	GET	All	Return a heartbeat.
/replay_buffer	GET	Learner	Ouput information in JSON format about the Reverb server.
/config	GET	Learner	Ouput the configuration file in JSON format.
/discovery	GET	Learner	Discover the services.
/model	POST	Actor, Evaluator	Sends the path of the newest version of the model.
/stats	GET	Actor, Evaluator	Manually get the statistics generated by the service.

Table 5.1: Supported routes

¹³Note that this is not the only way to do that, the implementer is free to make his own implementation of **Orchestrator**, **ReplayBuffer** and **ModelBroadcaster**

¹⁴Here node refers to a machine.

¹⁵Either actor or learner or evaluator

5.4 gRPC server

The learner service also deploys a **Reverb**¹⁶ server, which we will use for storing annotated games and sampling from them. It is based on gRPC Remote Procedure Calls (gRPC)¹⁷, and designed primarily for efficiency.

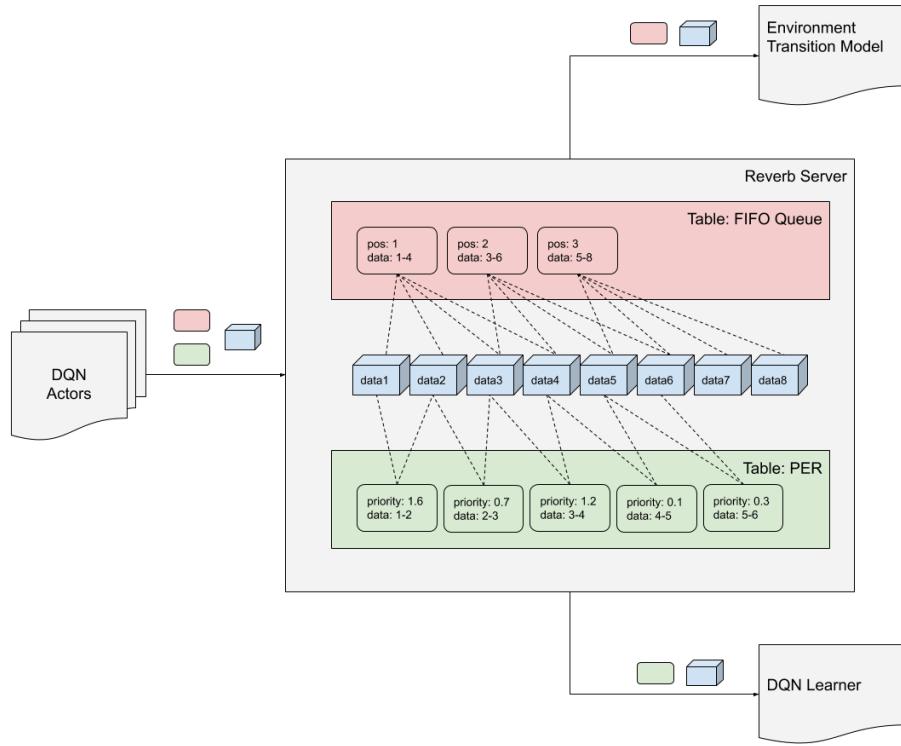


Figure 5.6: Illustration of how Reverb works

This server will act as an intermediate between the actors and the learner in the following sense:

- The implemented **Orchestrator** will send batches of generated games to the server.
- The implemented **ReplayBuffer** will take samples from the **Reverb** server for training purposes.

5.5 Tensorboard server

6 Implementation

7 Deployment

¹⁶Reverb is an efficient data storage and experience replay system for distributed reinforcement learning. It supports multiple data structure representations such as FIFO, LIFO, and priority queues.

¹⁷This is a recursive acronym.

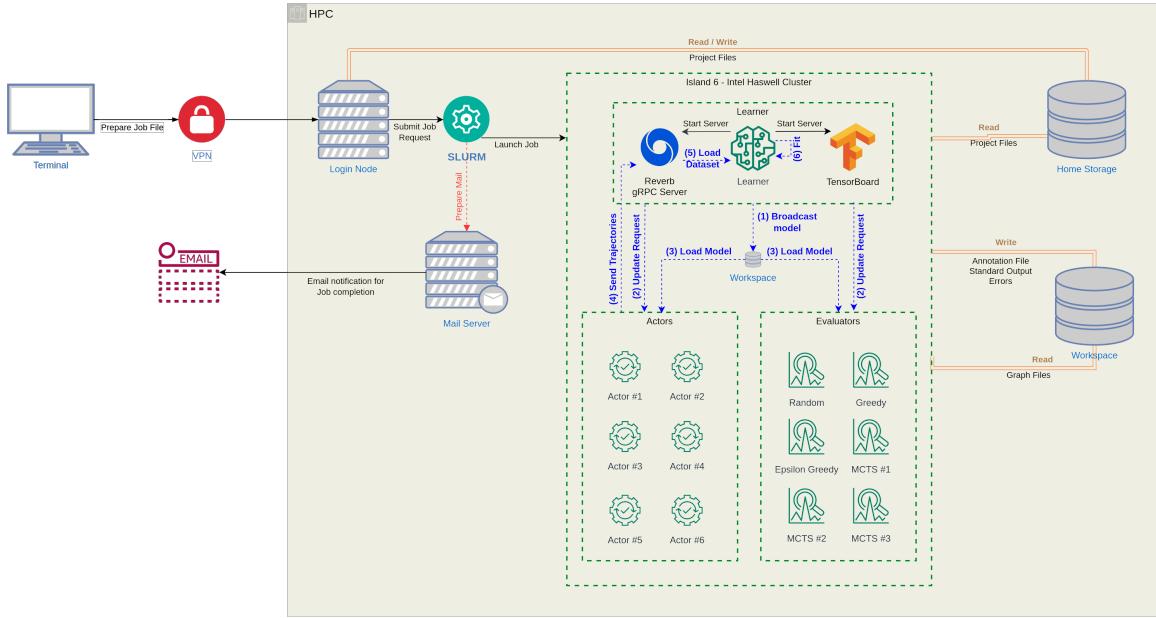


Figure 5.7: Pipeline Architecture

8 Configuration

We externalized the configuration of our self-play pipeline, so that we could change the parameters and fine-tune the hyper-parameters without modifying the source code¹⁸. This gives us the possibility to play experiments until getting the desired computational and prediction performances.

We created a YAML configuration file with the name `configuration.yml` that we already introduced in section 5. It will be used by all services¹⁹. When we completed the externalization process, we found out that we need also to set the file as a template so we can change it dynamically by environment variables. This is beneficial as we do have many instances of the same services that only differ by some minor hyper-parameters.

To achieve this, we introduced **Jinja** templating into the file, and set the name of the template configuration file as `configuration.yml.j2`. The templated configuration variables includes:

- **path**: The directory of the pipeline
- The instance name of the service.
- The port used by the services REST server.
- Type of the opponent for an evaluator service.
- The address and port of the gRPC service.

In the remaining of this section, we will describe the remaining configurations except the training and model configurations as they were already discussed in figures 4.11 and 4.12.

¹⁸Still, the pipeline needs to be restarted for changes to take effect.

¹⁹In a future iteration, we plan to split this file per service.

8.1 Replay buffer

Our pipeline supports two kinds of replay buffer. A “local” one, that we can use in a centralized system. And a “grpc” one²⁰, which is the one we are using.

The figure below shows the configuration parameters of the replay buffer:

```

replay_buffer:
  buffer_size: {{BUFFER_SIZE | default(1000000) }}
  reuse: 32
  implementation:
    table: "mpg"
    type: grpc #One of "local", "grpc"
    address: {{ GRPC_ADDRESS | default("auto") }} # Address of the replay buffer. Auto will automatically detect the address
    port : {{GRPC_PORT | default(50051) }}
    sampler:
      name: "random" # one of "random", "fifo", "lifo", "priority"
      params: [] # Additional parameters for the sampler
    remover:
      name: "fifo" # one of "random" or "fifo", "lifo", "priority"
      params: [] # Additional parameters for the remover
    min_size: {{BUFFER_MIN_SIZE | default(8192)}} # minimum number of elements in the buffer before sampling
    max_in_flight_samples_per_worker: {{MAX_FLIGHT_SAMPLES | default(192)}} # maximum number of samples that can be in flight per worker
  value_target: value # The objective of the game. One of "winner", "mean_payoff".
  # winner is the objective of determining the winner of the game
  # mean_payoff is the objective of determining the mean payoff of the game
  # value the mcts value of the state
  writer_sampler: 0.1 # If string, one of "trajectory", "random"

```

Figure 5.8: Replay buffer section of the configuration file

Note: while this is a recent version of the configuration file, we may tweak it further for fine-tuning purposes.

8.2 Services

We also externalized the parameters of the services, so we can easily change their deployment options. This is shown by the figure below:

²⁰Which is the **Reverb** server.

```

services:
  actors:
    instances: {{ACTOR_INSTANCES | default(6) }}
    port: {{ACTOR_PORT | default(13252)}} # Port to use for the actors
    stats_file: stats.jsonl # Base name of the file to store the statistics of the actors
    # The file will be named as <stats_basename>.<instance_id>.json
    stats_frequency: 100 # Frequency of the statistics collection in seconds
    max_collection_time: 100 # Maximum time to collect data in seconds. If null, data collection is run indefinitely.
    collection_period: 3 # Period between data collection in seconds. If 0, data collection is run only once.
    request_length: 64 # The number of samples to send in a single request.
    collection_probability: 0.01 # The probability of collecting data from an actor
  evaluators:
    instances: {{EVALUATOR_INSTANCES | default(3) }}
    port: {{EVALUATOR_PORT | default(13251)}} # Port to use for the evaluators
    stats_file: stats.jsonl # Base name of the file to store the statistics of the evaluators
    # The file will be named as <stats_basename>.<instance_id>.json
    opponent: {{EVALUATOR OPPONENT | default("mcts") }} # One of "random", "mcts", "greedy"
    stats_frequency: 100 # Frequency of the statistics collection in seconds
    evaluation_window: 100
    evaluation_levels: 7
    max_collection_time: 10 # Maximum time to collect data in seconds. If null, data collection is run indefinitely.
    collection_period: 3 # Period between data collection in seconds. If 0, data collection is run only once.

```

Figure 5.9: Replay buffer section of the configuration file

Note: while this is a recent version of the configuration file, we may tweak it further for fine-tuning purposes.

9 Execution

Conclusion

This chapter marks the culmination of all our research, proven results, implementations into a whole RL system based on SP.

First of all, we started by formalising our RL approach, we formalised our MPG as a SG consistently with the model we had built in chapter 4.

We also defined two additional MPG distributions that will be used on the SP pipeline. The choice of both distributions is consistent with both datasets discussed in chapter 3. This is intended so that we can evaluate our algorithms against optimal players.

Second of all, we formalised the MCTS approach, and followed Alpha Zero's approach to get a MCTS variant that uses the model defined in chapter 4.

Later, we separated functionalities of the SP pipeline following Alpha Zero's approach. Then we distributed the whole system so it can be scaled both horizontally and vertically.

Once we had resolved all implementation issues, we deployed the SP pipeline to a HPC cluster, externalized the configuration of the whole system.

Finally, we executed the whole SP pipeline as a proof of concept.

6

Analyse

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

$$\text{MA}(x, Y, c) \iff x \leq \max Y + c$$

New York (NY) Hi

Conclusion & Perspectives

In this internship, we

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 prédiction 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'exécution 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 mesure de coût d’entraînement. Une fois le problème de déploiement est résolu, nous recommanderions l’utilisation de ce même métrique pour estimer le coût d’interférence qui va justifier l’utilisation des BNNs.

A

On Constraint Satisfaction 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.

1 Constraint Satisfaction Problem

1.1 Definition

A constraint satisfaction problem

1.2 Assignment

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

1.3 Polymorphism

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

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

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

2 Ternary Max Atom Systems

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 $\text{MA}_3(x, y, z, c)$ be defined as follow:

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

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

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

2.2 Example

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

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

3 Max Atom Systems

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 $\text{MA}(x, Y, c)$ be defined as follow:

$$\text{MA}(x, Y, c) \iff x \leq \max Y + c$$

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

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

3.2 $\text{MA} \leq \text{MA}_3$

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

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

We will prove that S' is equivalent to S .

3.2.1.1 Implication

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

3.2.1.2 Equivalence

- Let $X : \mathcal{X}' \rightarrow D$ 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'

3.2.2 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 $\text{MA}(x, Y, c)$ with $x \in \mathcal{X}^*, Y \subseteq \mathcal{X}^*, c \in I$ with $|Y| \leq 2$

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 $R = \text{MA}(x, Y, c)$ we map it to the relation $R_3 = \text{MA}(x, y, z, c)$ 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 singleton¹ Y .

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

Algorithm 18 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

```

 $S' \leftarrow \emptyset$ 
 $H \leftarrow \emptyset$   $\triangleright H$  is a symmetric map between variable,variable to variables
 $V \leftarrow \text{Variables}(S)$   $\triangleright V$  is a set containing all variables
for  $\mathcal{C} \in S$  do  $\triangleright$  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
             $w \leftarrow \text{newVariable}(V)$   $\triangleright$  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  $S'$ 

```

3.3 Polymorphisms

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

- The max polymorphisms M^k defined by:

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

- The translation polymorphisms T_c defined by:

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

¹A set with only one element

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 $\text{MA}(x, Y, C)$ be defined as follow:

$$\text{MA}(x, Y, c) \iff x \leq \max(Y + C)$$

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

$$\text{MI}(x, Y, C) \iff x \leq \min(Y + C)$$

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

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

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 $S' = \text{CSP}(D, \Gamma)$ be a Min Max system, and let:

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

4.1.1 Transforming MI constraints

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

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

4.1.2 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:

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

4.1.3 Building the Max Atom System

Now, let:

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

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

4.1.4 Equivalence

Let:

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

If $\text{OP} = \text{MI}$, 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:

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

Now, we also have:

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

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

Algorithm 19 Converting a Min-Max System to Max Atom

Require: S a Min-Max system

Ensure: S' an N -ary Max Atom system

```

 $S' \leftarrow \emptyset$ 
 $H \leftarrow \emptyset$                                  $\triangleright H$  is a map between variable,offsets to variables
 $V \leftarrow \text{Variables}(S)$                      $\triangleright V$  is a set containing all variables
for  $\mathcal{C} \in S$  do                           $\triangleright$  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
         $Z \leftarrow \emptyset$ 
        for  $(y, c) \in \text{zip}(Y, C)$  do
            if  $(y, c) \notin \text{domain } H$  then
                 $z \leftarrow \text{newVariable}(V)$   $\triangleright$  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

```

5 Solving Mean Payoff

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].

We have made a slight modification, as the original equivalence solves the problem “Does Max wins² at all starting vertices?”, our version instead solves the problem “In which starting states Max wins at all vertices?”

Furthermore, the solution is constructive, in the sense that if the answer is positive, we can build an optimal³ strategy.

Algorithm 20 Converting a Mean Payoff Game to a Min Max system

Require: G a Mean Payoff Game

Ensure: S an Min-Max system

```

 $E \leftarrow E(G)$                                  $\triangleright$  The edges of  $G$ 
 $V \leftarrow V(G)$                                  $\triangleright$  The variables of  $G$ 
 $W \leftarrow W(G)$                                  $\triangleright$  The weight function of  $G$ 
 $P \leftarrow P(G)$                                  $\triangleright$  The set of player of  $G$ 
for  $(u, p) \in V \times P$  do
     $x \leftarrow (u, p)$ 
     $A \leftarrow \text{Adj}(x)$ 
     $Y = \{(a, \bar{p}), \quad a \in A\}$ 
     $C \leftarrow W(A)$                                  $\triangleright$  Calculating the weights element wise.
    if  $p$  is Max then:
         $OP \leftarrow MA$ 
    else
         $OP \leftarrow MI$ 
    end if
     $S \leftarrow S \cup \{OP(x, Y, C)\}$ 
end for

```

5.2 Arc Consistency

5.2.1 First Implementation

At first, we took the original implementation of AC3 [40, page. 171] and modify it to support the max atom system:

²Winning here is defined as condition C_1 in table 2.1

³Weakly-optimal with the C_1 winning condition.

Algorithm 21 AC3 for Ternary Max Atom systems

Require: \mathcal{C} a ternary Max Atom constraint
Require: $\nu : V \rightarrow \mathcal{P}(D)$ the admissible values function
Require: Q a Queue of pending updates
Ensure: ν update the 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 side constant  $\mathcal{C}$ 
 $Z \leftarrow [x, y, z]$ 
for  $o \in \{0, 1, 2\}$  do ▷ Iterate over all rotations
     $\text{admissible} \leftarrow \text{False}$ 
     $x \leftarrow Z[o]$ 
     $y \leftarrow Z[(o + 1) \bmod 3]$ 
     $z \leftarrow Z[(o + 2) \bmod 3]$ 
     $\mu \leftarrow \emptyset$  ▷ Set of values to delete
    for  $a \in \nu(x)$  do
        for  $(b, c) \in \nu(y) \times \nu(z)$  do
             $T \leftarrow [a, b, c]$ 
             $p \leftarrow T[(-o) \bmod 3]$ 
             $q \leftarrow T[(1 - o) \bmod 3]$ 
             $r \leftarrow T[(2 - o) \bmod 3]$ 
            if  $p \leq \max(q, r) + c$  then
                 $\text{admissible} \leftarrow \text{True}$ 
            end if
        end for
        if not  $\text{admissible}$  then
             $\mu \leftarrow \mu \cup \{a\}$ 
        end if
    end for
    if  $|\mu| > 0$  then
         $\nu(x) \leftarrow \nu(x) \setminus \mu$ 
         $\text{append}(Q, \text{Adj } x)$ 
    end if
end for

```

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

5.2.2 Refinement

We were able to make many simplifications to the algorithm by taking advantage of the symmetry⁴ of the system and the fact that it has translations and maximum as polymorphisms [6]:

Algorithm 22 AC3 Optimized for Ternary Max Atom systems

Require: \mathcal{C} a ternary Max Atom constraint

Require: $\nu : V \rightarrow D$ 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

⁴max is a symmetric function: $\max(x, y) = \max(y, x)$

B

On Random Mean Payoff Graphs

Introduction

This appendix deals with some random properties of MPG. The results offered by the appendix are used in the implementation and complexity analysis used in section 5 of chapter 3.

1 Sinkless $\mathcal{D}(n, p)$ Graph

1.1 Property

Let \mathbb{P} be the property¹ “Graph has not sink”.

This property is increasing in the sense that:

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

As a consequence:

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

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 $\text{Sinkless}(G)$.

¹Formally, a property is just a set of graphs. In practice, it is a set that has desirable “properties”.

1.2 Basic Comparison with Normal $\mathcal{D}(n, p)$

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

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

With that:

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

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 $\mathcal{D}(n, p)$

1.3 Property Probability

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

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

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

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:

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

1.4 Asymptotic Analysis For Dense $\mathcal{D}(n, p)$

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

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

Which implies:

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

If we take the limit, we have:

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

By tending c to 0, we get:

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

1.5 Asymptotic Analysis For Sparse $\mathcal{D}(n, p)$

Let:

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

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:

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

By applying the exponential function to both sides:

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

Now, we apply \ln to both sides, and multiply by x :

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

Finally, we apply the exponential function to both sides, to get the desired estimation of f :

$$f(x, k, x) = e^{-\frac{e^{-c}}{x^{k-1}} + o(\frac{1}{x^{k-1}})} \quad (\text{B.2})$$

Now with that:

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

Finally, we can conclude that:

$$\lim_{x \rightarrow +\infty} f(x, k) = \begin{cases} 0 & \text{if } k \in [0, 1[\\ e^{-e^{-c}} & \text{if } k = 1 \\ 1 & \text{otherwise if } k \in]1, +\infty[\end{cases} \quad (\text{B.3})$$

2 Verification of distribution properties

In section 4.1 of chapter 3, we required our MPG distributions to be fair and symmetric.

We have proposed some useful weight distributions and graph distributions in table 3.1. Then, we have also boldly claimed that any combination of both types of such distributions will give a MPG distribution verifying both properties.

In this section, we will prove that claim. We will first start by the following notations:

- Let \mathcal{D} be a graph distribution.
- Let \mathcal{W} be a symmetric weight distribution.
- Let $D = (V, E) \sim \mathcal{D}$, and let $n = |V|$
- Let $W \sim \mathcal{W}$
- Let $s \sim \mathcal{U}(V)$
- Let $p \sim \mathcal{U}(P)$ where $P = \{\text{Max}, \text{Min}\}$
- Let $G = (V, E, W, s, p)$

2.1 Symmetric

This is immediate as \mathcal{W} is symmetric by construction.

2.2 Fairness

$$\begin{aligned}
 \mathcal{P}(\text{Max wins in } G \text{ with optimal play}) &= \mathcal{P}(v(G) > 0) \\
 &= \mathcal{P}(v(\bar{G}) < 0) \\
 &= \mathcal{P}(v(V, E, -W, s, \bar{p}) < 0) \\
 &= \mathcal{P}(v(V, E, W, s, \bar{p}) < 0) \quad \text{as } W \sim \mathcal{W} \text{ and } \mathcal{W} \text{ is symmetric} \\
 &= \mathcal{P}(v(V, E, W, s, p) < 0) \quad \text{as } p \sim \mathcal{P} \\
 &= \mathcal{P}(v(G) < 0) \\
 &= \mathcal{P}(\text{Min wins in } G \text{ with optimal play})
 \end{aligned}$$

2.3 Conclusion

We conclude the result for any combination of distributions following the table 3.1 as they follow the hypotheses.

3 Expected Mean Payoff

3.1 Definition

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

3.2 Matrix Form

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

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

C

On Probabilistic Strategies

This appendix is dedicated to building important results related to fractional strategies.

1 Markovian Nature

1.1 Fixing Π^{Opt}

Let $G = (V, E, W, s, p)$ be a MPG.

If P 's is fixed to a fractional strategy Π^{Opt} , and $\bar{\text{Opt}}$ is still variable, then the SG becomes a MDP.

1.1.1 Transitions

This is illustrated as follow:

- We set the initial state of the MDP Γ as follow:

$$\begin{cases} s & \text{if } p = \bar{\text{Opt}} \\ \Pi^{\text{Opt}}(s) & \text{otherwise} \end{cases}$$

- If the current state is $u \in V$, then for any action $a \in V$, we have:

$$\mathcal{P}(u \rightarrow v \mid a) = \begin{cases} \mathcal{P}(\Pi^{\text{Opt}}(a) = v) & \text{if } a \in \text{Adj } v \\ 0 & \text{otherwise} \end{cases}$$

1.1.2 Rewards

We have:

$$\begin{aligned} \forall u, a, v \in V, \quad R(u, a, v) &= W(u, a) + W(a, v) \\ \forall u, a \in V, \quad R(u, a) &= R(u, a, \Pi^{\text{Opt}}(a)) \\ &= W(u, a) + \sum_{v \in \text{Adj } a} \mathcal{P}(\Pi^{\text{Opt}}(a) = v) \times W(a, v) \end{aligned}$$

If we represent R, W and Π^{Opt} as matrices, then:

$$R = W + \mathbf{1} \times (\Pi^{\text{Opt}} \odot W) \quad (\text{C.1})$$

1.2 Fixing both Π^{Max} and Π^{Min}

- Let $G = (V, E, W, s, p)$ be a MPG.
- Suppose without a loss of generality that $p = \text{Max}$

We have:

$$\begin{aligned} \forall u, v \in V, \quad \mathcal{P}(u \rightarrow v) &= \mathcal{P}(\Pi^{\text{Max}} \circ \Pi^{\text{Min}}(u) = v) \\ &= \sum_{w \in V} \mathcal{P}(\Pi^{\text{Min}} \circ \Pi^{\text{Max}}(u) = v \mid \Pi^{\text{Max}}(u) = w) \times \mathcal{P}(\Pi^{\text{Max}}(u) = w) \\ &= \sum_{w \in V} \mathcal{P}(\Pi^{\text{Min}}(w) = v) \times \mathcal{P}(\Pi^{\text{Max}}(u) = w) \end{aligned}$$

If we represent Π^{Max} and Π^{Min} as matrices, then we get the transition matrix of the Markov Chain:

$$M = \Pi^{\text{Max}} \times \Pi^{\text{Min}} \quad (\text{C.2})$$

2 Expected Reward of a MRP

2.1 Markov Reward Process

2.1.1 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
- $W : E \rightarrow \mathbb{R}$ is the weights function
- $A : E \rightarrow [0, 1]$ is the transition function, satifying:

$$\sum_{v \in \text{Adj } u} A(u, v) = 1 \quad (\text{C.3})$$

- $\gamma \in [0, 1]$ 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:

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

2.1.2 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 $\text{Adj } X_{n-1}$ using probabilities from A

The cumulative (discounted) reward¹ $R_\gamma(s)$ is defined as:

$$R_\gamma(s) = \sum_{n \in \mathbb{N}} \gamma^n W(X_n, X_{n+1}) \quad (\text{C.4})$$

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

For undiscounted rewards², such reward may not converge.

For that the average time reward $\bar{R}(s)$ starting from s , is defined as follow:

$$\bar{R}(s) = \lim_{n \rightarrow +\infty} \frac{1}{n} \sum_{k=0}^{n-1} W(X_k, X_{k+1}) \quad (\text{C.5})$$

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 $\gamma \in [0, 1[$.

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

Now, by considering S_n, A, W as matrices, the equation can be simplified to³ :

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

Here:

¹“reward” and “payoff” here as synonymous. We use the word “reward”

²Which means $\gamma = 1$.

³ γ has to be less then 1, as otherwise we cannot generally invert the matrix $I - \gamma A$.

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

With that, the discounted reward is calculated as follow:

$$\mathbb{E}[R_\gamma] = (I - \gamma A)^{-1}(A \odot W)\mathbf{1} \quad (\text{C.6})$$

2.3 Expected average-time reward

While the discounted 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.

2.3.1 Deriving Formula

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

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

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

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

$$\mathbb{E}[\bar{R}] = \lim_{n \rightarrow +\infty} \frac{1}{n} \sum_{k=0}^{n-1} A^k (A \odot W)\mathbf{1} \quad (\text{C.7})$$

2.3.2 Convergence

It is not trivial that the right-hand side of (C.7) converges.

Rothblum[39] proved that such limits exists and calculated its value. We will reformulate his results in the following theorem.

Theorem 7 For any stochastic matrix A , the limit $\lim_{n \rightarrow +\infty} \frac{1}{n} \sum_{k=0}^{n-1} A^k$ exists, and is equal to the **projection** matrix T **uniquely** defined as follow:

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

Now theorem 7 gives a straightforward construction of T . We only have to build a projection matrix such that:

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

Furthermore, Rothblum[39] gave a closed-form expression that applies to such projection T :

$$T = \lim_{n \rightarrow +\infty} \frac{1}{n} \sum_{k=0}^{n-1} A^k = X(Y^H X)^{-1} Y^H \quad (\text{C.8})$$

Where:

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

3 Evaluation of probabilistic strategies

3.1 Definition

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

3.2 Matrix Form

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

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

Bibliography

- [1] Dorit Aharonov and Oded Regev. “Lattice Problems in $\text{NP} \cap \text{coNP}$ ”. In: *Foundations of Computer Science, 1975., 16th Annual Symposium on* (May 2004). DOI: [10.1109/FOCS.2004.35](https://doi.org/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](https://doi.org/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](https://arxiv.org/abs/1308.3432). URL: <http://arxiv.org/abs/1308.3432>.
- [5] C. BERGE et al. “Contributions to the Theory of Games (AM-39), Volume III”. In: Princeton University Press, 1957, 179–187. ISBN: 9780691079363. URL: <http://www.jstor.org/stable/j.ctt1b9x26z> (visited on 09/03/2023).
- [6] Manuel Bodirsky and Marcello Mamino. *Tropically convex constraint satisfaction*. 2017. arXiv: [1506.04184 \[cs.CC\]](https://arxiv.org/abs/1506.04184).
- [7] Alper Kamil Bozkurt et al. *Model-Free Reinforcement Learning for Stochastic Games with Linear Temporal Logic Objectives*. 2020. arXiv: [2010.01050 \[cs.R0\]](https://arxiv.org/abs/2010.01050).
- [8] 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>.
- [9] 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](https://doi.org/10.1016/0890-5401(92)90048-K). URL: <https://www.sciencedirect.com/science/article/pii/089054019290048K>.
- [10] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), 273–297.
- [11] Rémi Coulom. “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search”. In: *Computers and Games*. Ed. by H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, 72–83. ISBN: 978-3-540-75538-8.
- [12] 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](https://arxiv.org/abs/1602.02830). URL: <http://arxiv.org/abs/1602.02830>.
- [13] 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](https://arxiv.org/abs/1511.00363). URL: <http://arxiv.org/abs/1511.00363>.
- [14] 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](https://doi.org/10.1007/BF01768705). URL: <https://doi.org/10.1007/BF01768705>.
- [15] Markus Enzenberger et al. “Fuego—An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 2 (2010), 259–270. URL: <https://api.semanticscholar.org/CorpusID:8323579>.
- [16] Alan Frieze and Michał Karoński. *Introduction to Random Graphs*. Cambridge University Press, 2015. DOI: [10.1017/CBO9781316339831](https://doi.org/10.1017/CBO9781316339831).
- [17] Elizabeth Gibney. “Self-taught AI is best yet at strategy game Go”. In: *Nature* (Oct. 2017). DOI: [10.1038/nature.2017.22858](https://doi.org/10.1038/nature.2017.22858).
- [18] 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>.
- [19] Julian Gutierrez, Thomas Steeples, and Michael Wooldridge. “Mean-Payoff Games with $\&\omega$ -Regular Specifications”. In: *Games* 13.1 (2022). ISSN: 2073-4336. DOI: [10.3390/g13010019](https://doi.org/10.3390/g13010019). URL: <https://www.mdpi.com/2073-4336/13/1/19>.
- [20] Koen Helwegen et al. “Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization”. In: *CoRR* abs/1906.02107 (2019). arXiv: [1906.02107](https://arxiv.org/abs/1906.02107). URL: <http://arxiv.org/abs/1906.02107>.
- [21] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), 1735–1780.
- [22] IBM. *Présentation générale de CRISP-DM*. <https://www.ibm.com/docs/fr/spss-modeler/saas?topic=dm-crisp-help-overview>. 2017.
- [23] 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>.
- [24] Zohar Jackson. *Free Spoken Digits Dataset*. <https://github.com/Jakobovski/free-spoken-digit-dataset>. 2019.
- [25] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).
- [26] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. USA: Addison Wesley Longman Publishing Co., Inc., 1997. ISBN: 0201896834.
- [27] Levente Kocsis and Csaba Szepesvári. “Bandit Based Monte-Carlo Planning”. In: *Machine Learning: ECML 2006*. Ed. by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, 282–293. ISBN: 978-3-540-46056-5.
- [28] 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/c399862d3b9d6b76c8436e924a68c9.pdf>.
- [29] 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](https://doi.org/10.1109/5.726791).

- [30] Yann LeCun. *THE MNIST DATABASE*. <http://yann.lecun.com/exdb/mnist>. 1998.
- [31] Xiaofan Lin, Cong Zhao, and Wei Pan. “Towards Accurate Binary Convolutional Neural Network”. In: *CoRR* abs/1711.11294 (2017). arXiv: [1711.11294](https://arxiv.org/abs/1711.11294). URL: <http://arxiv.org/abs/1711.11294>.
- [32] Seppo Linnainmaa. “Taylor Expansion of the Accumulated Rounding Error”. In: *BIT* 16.2 (1976), 146–160. ISSN: 0006-3835. DOI: [10.1007/BF01931367](https://doi.org/10.1007/BF01931367). URL: <https://doi.org/10.1007/BF01931367>.
- [33] 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](https://arxiv.org/abs/1808.00278). URL: <http://arxiv.org/abs/1808.00278>.
- [34] Kim Martineau. *Shrinking deep learning’s carbon footprint*. 7Aug 2021. URL: <https://news.mit.edu/2020/shrinking-deep-learning-carbon-footprint-0807>.
- [35] Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997.
- [36] *NP intersect coNP*. <https://kintali.wordpress.com/2010/06/06/np-intersect-conp/>. Accessed: 2023-08-28.
- [37] Nicolai Pisaruk. “Stochastic mean payoff games”. In: Apr. 2014.
- [38] Mohammad Rastegari et al. “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks”. In: *CoRR* abs/1603.05279 (2016). arXiv: [1603.05279](https://arxiv.org/abs/1603.05279). URL: <http://arxiv.org/abs/1603.05279>.
- [39] 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](https://doi.org/10.1007/BFb0120751). URL: <https://doi.org/10.1007/BFb0120751>.
- [40] 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>.
- [41] 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](https://www.sciencedirect.com/science/article/pii/S0167639311001622). URL: <https://www.sciencedirect.com/science/article/pii/S0167639311001622>.
- [42] 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](https://doi.org/10.23915/distill.00033).
- [43] 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.
- [44] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), 484–489. ISSN: 1476-4687. DOI: [10.1038/nature16961](https://doi.org/10.1038/nature16961). URL: <https://doi.org/10.1038/nature16961>.
- [45] 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](https://doi.org/10.1038/nature24270). URL: <https://doi.org/10.1038/nature24270>.

- [46] Eilon Solan and Nicolas Vieille. “Stochastic games”. In: *Proceedings of the National Academy of Sciences* 112.45 (2015), 13743–13746. DOI: [10.1073/pnas.1513508112](https://doi.org/10.1073/pnas.1513508112). eprint: <https://www.pnas.org/doi/pdf/10.1073/pnas.1513508112>. URL: <https://www.pnas.org/doi/abs/10.1073/pnas.1513508112>.
- [47] 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](https://arxiv.org/abs/1906.02243). URL: [http://arxiv.org/abs/1906.02243](https://arxiv.org/abs/1906.02243).
- [48] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.
- [49] Tim Dettmers. *Deep Learning in a Nutshell: History and Training*. <https://developer.nvidia.com/blog/deep-learning-nutshell-history-training/>. 16 Dec. 2015.
- [50] 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](https://doi.org/10.1007/s43681-021-00043-6).
- [51] Chunyu Yuan and Sos S. Agaian. “A comprehensive review of Binary Neural Network”. In: *CoRR* abs/2110.06804 (2021). arXiv: [2110.06804](https://arxiv.org/abs/2110.06804). URL: <https://arxiv.org/abs/2110.06804>.
- [52] Chunyu Yuan and Sos S. Agaian. “A comprehensive review of Binary Neural Network”. In: *CoRR* abs/2110.06804 (2021), 3–4. arXiv: [2110.06804](https://arxiv.org/abs/2110.06804). URL: <https://arxiv.org/abs/2110.06804>.
- [53] 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.
- [54] 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](https://doi.org/10.1007/s10462-022-10228-y). URL: <https://doi.org/10.1007/s10462-022-10228-y>.