POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

A Neural Network-Embedded Optimization Approach for Selecting Multiple Entries for *March Madness*

JEFF SYLVESTRE-DÉCARY
Département de mathématiques et de génie industriel

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

Août 2020

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

A Neural Network-Embedded Optimization Approach for Selecting Multiple Entries for *March Madness*

présenté par **Jeff SYLVESTRE-DÉCARY**

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

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

ACKNOWLEDGEMENTS

I wish to thank my supervisor Andrea Lodi for taking me under his supervision two years ago. For many years already, I have been highly stimulated by the idea of conducting research on sports. I mentioned it to him without any expectation, and he fulfilled my desire as quick as possible by reaching out to my co-supervisor David Bergman. I thank both of them to have make this dream of mine possible! Together, I felt like I had the perfect combination of advisor. Even though they both are extremely busy, I always felt they had time for me whenever I needed it! I especially enjoyed discussing new ideas with them, but they always make sure that the project was progressing in the right direction. David also introduced me to Jason who has also made himself available to discuss with me and have continually shared his thoughts on the direction of the project. I want to thank to all three of them for offering me this opportunity to discover the field of optimization, and for their careful proof-reading of this thesis. I am looking forward to continue collaborating with these three outstanding individuals!

I also want to thank all the members of the chair. Through the sport sessions, the lunch and the diner, I discovered in them excellent researchers, but more importantly amazing friends! I particularly want to thank all the members that crossed path with me in the master's room. Many of you became true friends that I know I will be able to count on for many years to come!

J'aimerais également remercier les francophones du groupe de recherche. Je me suis souvent tourné vers vous lorsque j'ai eu besoin d'aide, et vous m'avez toujours accueilli les bras grands ouverts. Plus particulièrement, je remercie Mehdi pour son soutient dans la progression de ma maîtrise, Khalid pour les discussions intéressantes et pour son aide cruciale pour régler tous mes problèmes informatiques, et Didier pour ses judicieux conseils tout au long de ma maîtrise.

Finalement, je suis à jamais reconnaissant de ma famille. Ils me soutiennent dans tous mes projets depuis toujours, et ils m'ont toujours permis de faire ce que j'aime dans la vie! Je remercie également mes amis pour leur soutien moral et leurs efforts constants à me changer les idées!

RÉSUMÉ

On peut s'attendre à une croissance en popularité des paris sportifs dans le marché américain suite à la légalisation de ceux-ci dans plusieurs états depuis 2018 [1]. De plus, l'augmentation de la quantité de données sur le sport et le développement de nouvelles métriques de performance sportive ont permis depuis quelques années d'avoir une approche statistique pour les problèmes de prise de décision dans le sport. Alors que la littérature sur les paris sportifs couvrent majoritairement des modèles probabilistes pour prédire le résultat d'un évènement, cette thèse s'intéresse plutôt au développement d'une stratégie optimale pour remporter un paris sportif, plus particulièrement le *Tournament challenge* tenu annuellement par *ESPN*.

Le *Tournament Challenge* demande aux participants de choisir le gagnant de chacune des 63 parties du *March Madness*, soit le championnat de fin de saison de basketball collégial américain. Il existe 2^{63} façons de sélectionner les gagnants du tournois. De plus, plusieurs millions de personnes y participent à chaque année. Généralement, seulement un petit pourcentage des meilleurs scores font un gain monétaire ce qui implique qu'un participant doit obtenir un meilleur score que plusieurs millions de personnes pour remporter un gain.

Kaplan et al. (2001) ont été les premiers à introduire une approche exacte qui maximise l'espérance de point produit par une entrée. Notre stratégie est la première à considérer plusieurs entrées dépendantes au *Tournament Challenge*. Notre stratégie cherche à maximiser l'espérance de points produit par le score maximal des k entrées. Deux problèmes découlent de cette stratégie, soit comment évaluer et comment optimiser la fonction objective. Nous présentons trois approches pour évaluer la fonction objective. Cela inclue une méthode exacte qui est un algorithme basé sur un arbre de décision et deux modèles approximatifs, soit une approche par simulation et une approche par apprentissage machine. À partir de ces différents modèles, nous développons deux heuristiques permettants d'optimiser la fonction objective, soit un algorithme génétique et un réseau de neuronnes intégré à un modèle en nombre entier. Finalement, nous comparons l'espérence de points produits ainsi que le vrai score obtenu par chacune des méthodes pour chaque tournois depuis 2002. Nos deux modèles surpassent pour chaque instance la solution optimal du modèle exacte avec une entrée.

ABSTRACT

Sports gambling are expected to grow in popularity in the US as they have been legalized by many states in the last two years [1]. The availability of sports data and the development of new metrics to evaluate the performance of either athletes or teams have allowed the use of statistical approaches to tackle decision-making problems in sports. While most papers in the literature investigate how to predict the outcome of a game, this thesis addresses the development of an optimal strategy to win a sports betting contest. Specifically, we focus on the *ESPN Tournament Challenge* which is a sport betting contest on the season-ending championship tournaments of americain college basketball, also known as the *March Madness*.

The *ESPN Tournament Challenge* asks participants to pick the winner of each of the 63 games in the *March Madness*. Thus, there is a total of 2^{63} different ways of filling the tournament which makes the challenge a complex task. Every year, millions of people aim to predict accurately the *March Madness*. This contest often adopts a top-heavy payoff structure which implies that a single participant needs to beat millions of participant to receive a positive payoff.

Kaplan et al. (2001) first introduce an exact approach to the problem by selecting a single-entry that maximizes the expected score. We propose a novel strategy that considers a multi-entry approach to the *Tournament Challenge*. Such a strategy maximizes the expected score of the maximum scoring entry. We face two main challenge, namely, (1) how to evaluate the objective function and (2) how to optimize it. We then present three approaches for the evaluation of the objective function. This includes an exact approach in a Tree-based algorithm and two approximate models, a simulation approach and a neural network approach. Based on these three different models to evaluate the objective function, we develop both a genetic algorithm and a neural network-embedded algorithm. Finally, we compare the expected score and the empirical score by each approach on each tournament played since 2002. Computational experiments show that the proposed models clearly outperform the single-entry exact approach on every instance.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii

RÉSUMÉ iv

ABSTRACT v

TABLE OF CONTENTS vi

LIST OF TABLES viii

LIST OF FIGURES ix

LIST OF SYMBOLS AND ACRONYMS x

CHAPTER 1 INTRODUCTION 1
- 1.1 Introduction 1
- 1.2 Problem description and preliminaries 3
- 1.3 Example of the calculation of the expectation of the maximum of two entries 5
- 1.4 Research objectives 6
- 1.5 Outline 7

CHAPTER 2 LITERATURE REVIEW 8
- 2.1 Multi-entry sports gambling 8
- 2.2 Discrete optimization 9
  - 2.2.1 Convex optimization 9
  - 2.2.2 Integer linear program 11
- 2.3 Machine learning 14
  - 2.3.1 Supervised learning 15
- 2.4 Embedding NNs in optimization (`JANOS`) 21
  - 2.4.1 NN with Rectified Linear Unit Activation Function in Janos 22
- 2.5 Optimizing the expected value of order statistics 24
- 2.6 Optimization for *March Madness* 25

CHAPTER 3 METHODOLOGY 29
  - 3.0.1 Exact approach 29
  - 3.0.2 Simulation approach 31

3.0.3 Approximation via NNs 32
3.1 Optimization algorithms 33
3.1.1 Genetic algorithm 33
3.1.2 NN-Embedded optimization via `JANOS` 33

CHAPTER 4 EXPERIMENTAL EVALUATION 36
4.1 Computational platform 36
4.2 Probability models 36
4.3 Calculation the Expected Value of the Maximum 37
4.4 Optimization results 39
4.4.1 Single-entry evaluation 39
4.4.2 Multi-entry evaluation 41

CHAPTER 5 CONCLUSION, FUTURE WORK AND RECOMMENDATIONS . . 47
5.1 Summary of work 47
5.2 Limitations 47
5.3 Future research 48

REFERENCES 49

APPENDICES 53

LIST OF TABLES

Table 1.1 Possible tournament results 5
Table 3.1 Tree-based algorithm branching rules 30
Table 4.1 **P1** model performance for year's 2002 to 2019 37
Table 4.2 Accuracy of NN for estimating the expectation of one entry . . 40
Table 4.3 `JANOS` solution quality for single-entry problem 41
Table 4.4 Accuracy of NN for estimating $E(\max_{k=1....K} S(B_k))$ 43
Table 4.5 Comparison of quality of solutions between `JANOS` and `GA` . . . 44
Table 4.6 Actual score obtained by every entry in the men's tournaments 45
Table 4.7 Actual score obtained by every entry in the women's tournaments 46
Table B.1 Expected score of $\mathtt{IP}^{\mathtt{sim-2}}$ and $\mathtt{SIP}^{\mathtt{k}}$ for the men's tournament . 55
Table B.2 Actual score of $\mathtt{IP}^{\mathtt{sim-2}}$ and $\mathtt{SIP}^{\mathtt{k}}$ for the men's tournament . . . 56

LIST OF FIGURES

Figure 1.1 2019 *March Madness* 2
Figure 1.2 Four-team tournament 5
Figure 2.1 Convex analysis 10
Figure 2.2 Simplex method 11
Figure 2.3 Cutting plane algorithm 13
Figure 2.4 Overfitting vs underfitting 17
Figure 2.5 Neural network structure 20
Figure 2.6 ReLU activation function 23
Figure 3.1 Tree-based algorithm example 29
Figure 3.2 Genetic algorithm 34
Figure 4.1 SAA vs Tree-based algorithm 38
Figure 4.2 Time (seconds) to solve each instances for the men tourneys . 42

LIST OF SYMBOLS AND ACRONYMS

ML	Machine learning
NCAA	National Collegiate Athletic Association
MILP	Mixed-integer linear programming
IP	Integer programming
LP	Linear programming
DO	Discrete optimization
NN	Neural network
DFS	Daily fantasy sports
$\texttt{GA}$	Genetic algorithm
MSE	Mean squared error
MLE	Maximum likelihood estimation
ROC	Receiver operating characteristic
AUC	Area under the curve
ReLU	Rectified linear activation function
SGD	Stochastic gradient descent
SAA	Sample average approximation
MAPE	Mean absolute percentage error
NN	Neural network model
$\texttt{IP}^{\texttt{MM}}$	IP model
$I(\cdot)$	Identity function
$\texttt{IP}^{\texttt{sim-2}}$	Integer model using the simulation model
$\texttt{SIP}^{\texttt{k}}$	Sequential integer programming model

CHAPTER 1 INTRODUCTION

1.1 Introduction

Betting on sports continues to grow in the U.S. and abroad. Many U.S. states legalized gambling following a 2018 Supreme Court ruling [1] that rescinded the federal government's ban on such activity. Within this newly legalized and regulated environment, U.S. states reported over $13 billion wagered at licensed sportsbooks in 2019 alone [2]. Globally, between both licensed and unlicensed operators, the database company Statista estimates that over $120 billion in U.S. dollars is gambled on sports every year [3].

In traditional sports gambling, bettors risk an amount of their choosing on the outcome of a single game by paying a wager. If their chosen team wins, they win back their wager, plus the amount of the wager less a cut taken by the sportsbook (sometimes called the "vig," this is usually around 9% of the wager). If their chosen team loses, they lose the wager.

Betting pools are a separate type of sports gambling, and form a significant subset of the overall sports gambling market. In betting pools, all participants pay an entry fee for the competition. The winning teams for many games have to be chosen by the participants, and the participant who correctly picks the most winners wins the pool of money (the operator of the pool may keep a small amount as payment).[1] By far the most popular form of betting pool is the annual National Collegiate Athletic Association (NCAA) men's basketball tournament that takes place every March. A survey conducted by Morning Consult and produced by the American Gaming Association showed that an estimated $4.6 billion would be spent by Americans on such pools in 2019 [4].

The NCAA basketball tournament, often termed *March Madness*, consists of 68 teams competing in a bracket-style, a single-elimination tournament. In the preliminary round, eight of the teams play in four separate games, with the winning teams advancing to the group of 64 that constitute the initial set up of the tournament. The tournament winner has to win six consecutive games to claim the championship. An example bracket from the 2019 tournament is shown in Figure 1.1 [5].

There are four "regions" (as designated by where in the U.S. the first two rounds of games are played), each with 16 teams seeded from 1-16 where seed 1 is the strongest team and seed 16

[1]Depending on the circumstances, betting pools can require participants to make all of their selections at once, or in an iterative fashion. Additionally, other scoring systems are possible (e.g., picking the winner of a designated *important* game is worth more than picking the winner of another game). Finally, pools can allow for a cascading payment system instead of the "winner-take-all" design.

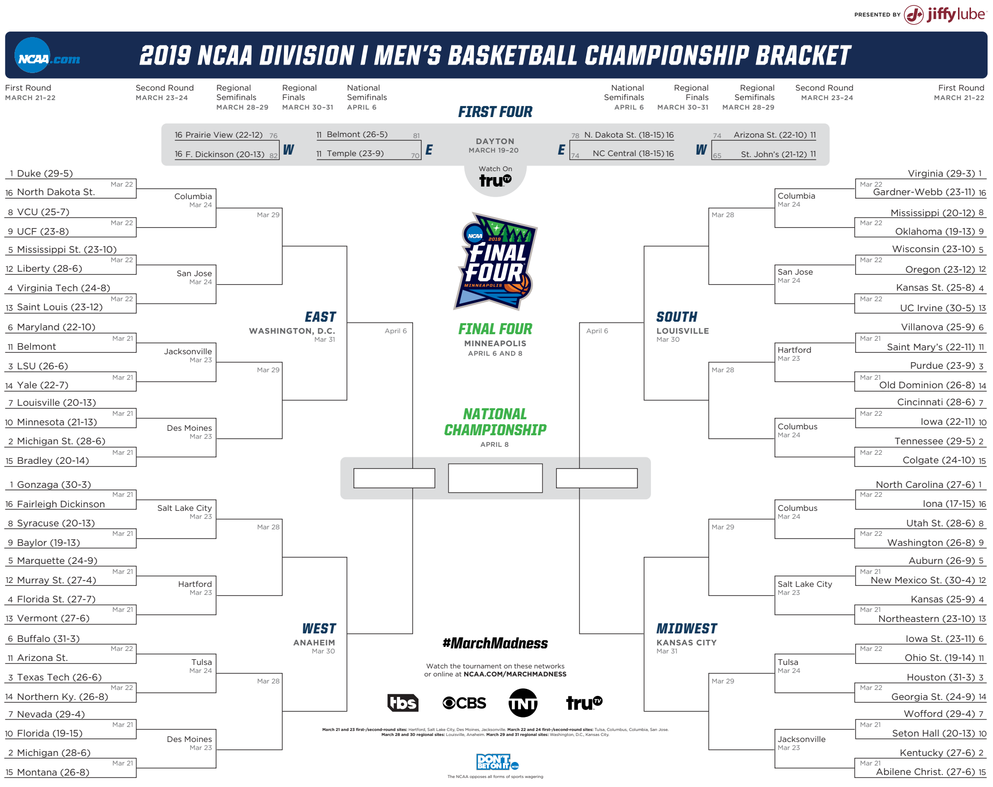


Figure 1.1 2019 *March Madness*

is the weakest team in the "region". Within each region, in the first round, seed numbers $n = (1, 2, 3, ..., 8)$ play against seed numbers $17 - n$. At no point is the tournament "reseeded" after the start. The four regional bracket winners advance to the stage of the tournament known as the *Final Four*, where the two semifinal games and then the championship game determine the winning team.

Popular interest in *March Madness* has been widespread for years. In 2019, over 40 million people filled out as estimated 149 million brackets [4]. Warren Buffett famously offered \$1 billion in 2014 to anyone filling out a perfect bracket [6], an offer that has since been changed to \$1 million a year for life to any of his employees for correctly picking all games in the first two rounds (48 games) [7]. And as popular interest has grown, so has academic interest. Kaggle has hosted annual machine learning competitions [8] from 2014-2020 in which participants competed to pick the best brackets each year using novel machine learning and optimization techniques.

Many papers focus on the idea of developing an accurate probabilistic model and on finding the best single-entry solution (e.g., [9], [10], [11], [12], [13], and [14]). Metrick [15] first presented evidence that participants in *March Madness* betting pools over-pick the heaviest favorites, suggesting that an understanding of the selection behavior of other participants can be just as important as picking winners of games correctly. In an analysis of over 200,000 brackets selected by participants in online pools in 2015 and 2016, Wright et al. [16] confirmed this tendency to over-back the top-ranked team. To our knowledge, none of the previous research in *March Madness* betting pools has focused on the possibility of submitting multiple bracket entries. Considering the aforementioned estimates that in 2019, 40 million people filled out 149 million brackets (approximately 3.7 brackets per person), a strategy for multi-entry participation is needed. The novelty of this thesis lies in presenting two heuristics that find a near-optimal solution (given a probabilistic model for game outcomes) to the problem of selecting multiple entries for a *March Madness* betting pool.

1.2 Problem description and preliminaries

A tournament $\mathcal{T} = \langle R, T, G\rangle$ is defined by the collection of rounds $R = \{1, \dots, |R|\}$ indexed by r, the collection of teams $T = \{1, \dots, |T|\}$ indexed by t, and the collection of games $G = \{1, \dots, |G|\}$ indexed by g. $\mathcal{T}$ is a single-elimination sport tournament with $|G| = 2^{|R|} - 1$ games and $|T| = 2^{|R|}$ teams. P is a $|T| \times |T|$ matrix defining the win probabilities for each team in every possible game. For $1 \leq t_i, t_j \leq |T|$, p_{t_i,t_j} in P is the probability that team t_i wins a game against team t_j (since no tie games can occur, $p_{t_i,t_j} + p_{t_j,t_i} = 1$). Round $r = 1$ features games between teams $2 \cdot g - 1$ and $2 \cdot g$ for $g = 1, \dots, 2^{|R|-1}$.[2] Iteratively from there, round r consists of $2^{|R|-r}$ games between a pair of teams, where for $g = 2^{|R|-r-1}+1, \dots, 2^{|R|-r}$, one is from the set $\{2^r \cdot (g - 2^{|R|-r-1} - 1) + 1, \dots, 2^r \cdot (g - 2^{|R|-r-1} - 0.5)\}$ and one from the set $\{2^r \cdot (g - 2^{|R|-r-1} - 0.5) + 1, \dots, 2^r \cdot g - 2^{|R|-r-1}\}$. The team from the corresponding sets is the sole team that won each of its games before round r. As an example, suppose $|R| = 3$ and we are considering round $r = 2$. This round will feature $2^{3-2} = 2$ games. One of the games will be between a team from $\{1, 2\}$ and a team from $\{3, 4\}$, and the other will be between a team from $\{5, 6\}$ and $\{7, 8\}$. Finally, let $T(g)$ be the collection of teams that might play in game g, $R(g)$ be the round of game g and $G(t, r)$ be the game that team t might play in round r.

A *March Madness* betting pool challenge consists of participants that select an outcome of every game for the entire tournament, before it starts. A single entry or a complete bracket $b : G \to T$ is an assignment of a team $t \in T$ to each game $g \in G$ where $t \in T(g)$. A single entry therefore selects a single team for each game which is understood to be selected as the

[2]This numbering is *not* the same as the "seeding" for the *March Madness* tournament described previously in the thesis and shown in Figure 1.1.

winner of the game. Note that the term "bracket" both in this thesis and colloquially can describe the outcome of the tournament, an entry in the pool, or a simulation of the outcome.

A *partial bracket* $\mathcal{B}_{w_d} : G \to T \cup \{\emptyset\}$ is a complete bracket but with the additional possibility of assigning the empty set to a game, which indicates that no team has been chosen as the winner of the game. Let $\mathcal{B}_{w_d=t}$ represent the subset of brackets in $\mathcal{B}$ with team t winning game d. We assume that both complete and partial brackets must be consistent in that if there is a team t' and a game g' for which $b[g'] = t'$ then t' must also be the winner in each game g for which $g \in G(t, r')\forall r' \in \{1, \ldots, R(g') - 1\}$ and $b[g] \leq b[g']$. Let $\mathcal{B}$ be the set of all complete brackets.

The tournament outcome is uncertain and will be a complete bracket $b^* \in \mathcal{B}$. Any entry $b \in \mathcal{B}$ earns points corresponding to a linear function of the number of correct assignments of winners in b, that are also winners in b^*. In this thesis we focus on the *ESPN Tournament Challenge*, where a game in round $r \in R$ is worth 2^{r-1} points and the *value* of bracket b given that b^* is the tournament outcome is denoted and defined by

$$S\left(b, b^*\right) := \sum_{g \in G : b[g] = b^*[g]} 2^{R(g)-1}.$$

The bracket with the maximum value is b^* and has value $v(b^*, b^*) = r \cdot 2^{r-1}$. Moreover, since the tournament outcome is uncertain, the score of a single-entry b is a random variable denoted as $S(b)$ and its expected score is denoted and defined by

$$E(S\left(b\right)) := \sum_{b^* \in \mathcal{B}} P_{b^*} \sum_{g \in G : b[g] = b^*[g]} 2^{R(g)-1}$$

where P_{b^*} is the probability that the bracket b^* occurs. P_{b^*} is calculated as

$$P_{b^*} = \prod_{\substack{g \in G: \\ t_i = b^*[g], \\ t_j = o(b^*, t_i, R(g))}} p_{t_i, t_j}$$

where $o(b^*, t_i, R(g))$ denote the opponent of team t_i in round $R(g)$ in the bracket b^*. Finally, let $S'(b, \mathcal{B}_{w_d})$ be the random variable of the score of bracket b when the partial bracket $\mathcal{B}_{w_d}$ is observed while, and $P_{\mathcal{B}_{w_d}}$ be the probability of observing the partial bracket $\mathcal{B}_{w_d}$.

1.3 Example of the calculation of the expectation of the maximum of two entries

The model presented in the previous section exhibits how difficult it is to optimize for the maximum expected score among a collection of k brackets. There might be an analytical formula to calculate the expected score of the maximum of k brackets, but we were not able to identify one.

Consider the following small tournament:

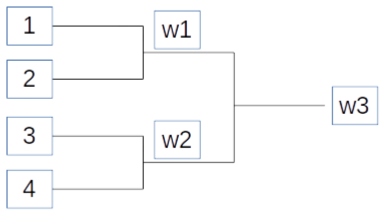


Figure 1.2 Four-team tournament

There's a total of two rounds and three games in this four-team tournament. There are $2^3 = 8$ feasible outcomes all listed in Table 1.1, where the winner for each game is listed in each column.

Table 1.1 Possible tournament results

$\mathcal{B}$	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
w_1	1	1	1	1	2	2	2	2
w_2	3	4	3	4	3	4	3	4
w_3	1	1	3	4	2	2	3	4

Assuming we select entry b_1 and entry b_6, here's an analytic formula to compute the expected score of the maximum of two entries.

$$\begin{aligned}E(\max(S(b_1, b_6)) &= \sum_{b_i \in \mathcal{B}} P_{b_i} \max(S(b_1, b_i), S(b_6, b_i)) \\ &= P_{b_1} \cdot \max(4,0) + P_{b_2} \cdot \max(3,1) + P_{b_3} \cdot \max(2,0) + P_{b_4} \cdot \max(1,1) + \\ &\quad P_{b_5} \cdot \max(1,3) + P_{b_6} \cdot \max(0,4) + P_{b_7} \cdot \max(1,1) + P_{b_8} \cdot \max(0,2) \\ &= P_{1,2} \cdot P_{3,4} \cdot P_{1,3} \cdot 4 + P_{1,2} \cdot P_{4,3} \cdot P_{1,4} \cdot 3 + P_{1,2} \cdot P_{3,4} \cdot P_{3,1} \cdot 2 + P_{1,2} \cdot P_{4,3} \cdot P_{4,1} \cdot 1 + \\ &\quad P_{2,1} \cdot P_{3,4} \cdot P_{2,3} \cdot 3 + P_{2,1} \cdot P_{4,3} \cdot P_{2,4} \cdot 4 + P_{2,1} \cdot P_{3,4} \cdot P_{3,2} \cdot 1 + P_{2,1} \cdot P_{4,3} \cdot P_{4,2} \cdot 2\end{aligned} \tag{1.1}$$

Generalizing this formula even in this small tournament appears to be particularly difficult, let alone for a larger tournament like *March Madness*. In addition, this only accounts for evaluating the quality of a solution, let alone identifying the optimal solution.

1.4 Research objectives

The goal of this thesis is to explore how a participant can select multiple entries in the *March Madness* pool. Specifically we seek to answer the following question: How can a participant select multiple entries in a *March Madness* pool to maximize the expected value of the maximum scoring entry?

The challenge of this problem lies in how to evaluate a solution (i.e., calculate the expectation of the maximum score of a collection of entries), in addition to devising a decision-making framework that allows us to search through all subsets of brackets and find the optimal collection. The approach we explore inherently requires us to model the dependency between solutions. As discussed in Section 2, there exist methods for maximizing the expected value for a single-entry selection problem, but the optimal single-entry bracket is not necessarily one of the brackets in an optimal solution to this multiple entries problem, necessitating reasoning over multiple brackets simultaneously.

As discussed in detail in this thesis, the evaluation of the expectation of the maximum of multiple entries is quite challenging. To the best of our knowledge, there is no analytical closed-form expression, even for just two entries. Our contributions in attacking this complicated problem are the following:

1. A tree-based dynamic programming model for evaluating exactly the expected value of the maximum score of a collection of brackets which can be stopped at any point to provide bounds

2. A genetic algorithm for finding a collection of brackets relying on simulation to evaluate the quality of the solution

3. A neural network-embedded IP model for finding a collection of brackets.

For the last point, we use a recently introduced solver called `JANOS` that allows users to pre-train neural network models and embed them in optimization model. This allows us to train a neural network to evaluate the quality of a collection of brackets and then, through `JANOS`, select proper brackets to maximize the output of the neural network.

1.5 Outline

We will begin this thesis with a literature review on multi-entry models of sports betting pools and single-entry approaches appearing in the literature for the *March Madness* tournament in Chapter 2. We will also introduce in this chapter the optimization and machine machine learning material required for the rest of the thesis. Chapter 3 will detail the technical contributions on this thesis. It starts with presenting three approaches for evaluating the expectation of the maximum scoring bracket out of a collection of brackets. The first is an exact approach, and the latter two are approximations, one using simulation and the other using neural networks. We then discuss two optimization algorithms for the multi-entry problem, both heuristics; a genetic algorithm and a neural network-embedded IP approach. Chapter 4 will detail the results from our experimental evaluation. The thesis will then conclude with a discussion of the approaches taken to solve this complex decision-making problem.

CHAPTER 2 LITERATURE REVIEW

2.1 Multi-entry sports gambling

The study of the selection of multiple entries in sports gambling competitions has gained popularity in the last few years. Bergman et al. [17] introduce a multi-entry strategy using stochastic programming to win a National Football League (NFL) survivor pool. NFL survivor pools require participant to select a team every week, and if the selected team wins its match, the participant advances to the following week. Every participant needs to select a team prior to the games playing in every week with the only constraint that a team can not be selected more than once by any entry. The winners of the pool are all the participants who survived the 17 weeks of the season, or last as long as they can. Their methodology utilizes integer programming to maximize the probability of surviving, and they show that maximizing the survival probability through a half-season look-ahead performs better than maximizing the survival probability over the entire season. Their model reports a 0.5 probability of winning a large contest at least one time in the next 30 years, an extremely profitable outcome.

Hunter et al. [18] and Haugh et al.[19] both tackle the multi-entry problem in Daily Fantasy Sports (DFS). DFS contests occur every day with scores for participants based on the performance of players selected sports competitions. A DFS contest requires participants to select a collection of players according to selection constraints. Each participant then receives points based on the performance of each player they select, and the winner is the participant who accumulates the most points after all matches are played. This presents a very difficult combinatorial problem in a similar fashion to *March Madness* pools because of the number of way of selecting all the players and the number of participants entering the contest.

Hunter et al. [18] selects entries by using a sequential integer programming (SIP) model. Their model selects entries that produce the highest expected score individually. Every time they select an entry, they sequentially add constraints that force the newly selected entries to have at most some number of identical players with all previously selected entries, leading to diversified entries that increase the variance of the score. Their best entry among 200 entries ranked in the top-3 best selected entries over ten weeks three different times.

Clair et al. [20] shows that the optimal strategy in a betting contest with a small number of opponents can be reduced to finding the solution that maximizes the expected score, but this strategy changes as the number of opponents increases. In large contests such as *March*

Madness or DFS, they demonstrate that many entries tend to be over-selected, leading to great opportunity to increase expected payoff by choosing under-selected entries even if their expected scores might be lower. With this concept in mind, Haugh et al.[19] introduce a model to simulate the opponents' selection. This model is used to establish a score to beat in order to be *in-the-money.* Utilizing mean-variance theory, their objective function maximizes the probability of exceeding the stochastic benchmark. To extend their formulation to the multi-entry problem, they use a SIP model in a similar fashion to Hunter et al. [18]. The model of Haugh et al.[19] outperforms by three times profit and loss over the benchmark they use for evaluation over 17 weeks of action in NFL DFS contest.
Bergman et al. [17] methodology happens to be too different to implement in a *March Madness* pool, but Hunter et al. [18] and Haugh et al.[19] could both be adapted to this contest. In Appendix B, we introduce a SIP for the *March Madness* pools, but this dissertation does not take in consideration what other participants select as in Clair et al. [20] and Haugh et al.[19]. Nonetheless, this thesis focused on developing methodology for selection a collection of entries simultaneously that maximize the expected score of the maximum scoring entry, rather than sequentially finding good entries. This task is more complicated than a simple sequential optimization model, because it directly incorporates reasoning based on the stochastic dependency between the entries.

2.2 Discrete optimization

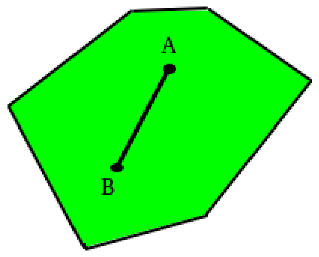
This section is based on the book Integer Programming by Wolsey [21].

Discrete optimization (DO) is a subclass of optimization (or mathematical programming) that aims at maximizing or minimizing a given function which depends on a discrete set. A discrete set can either be an integer program for which the variables are constrained to be integers or a combinatorial problem where the algorithm is required to find the discrete set of objects. DO problems usually face a large number of feasible solutions up to the point where it is impossible to find the optimal solution by exhaustive search. Due to the wide range of applications of DO, there has been a big amount of research in the past 50 years leading to advanced commercial solvers. In this section, we will go over the concepts that form the foundation of DO and commercial solvers.

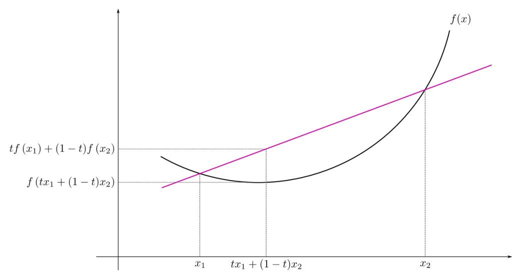
2.2.1 Convex optimization

C is a convex set (See Figure 2.1 (a)) if and only if

$$\forall\, x, y \in C, \forall\, t \in [0,1] : (1-t)x + ty \in C\ .$$



(a) Convex set



(b) Convex function

Figure 2.1 Convex analysis

Similarly, a function $f(\cdot)$ is convex (See Figure 2.1 (b) [22]) if and only if

$$\forall x_1, x_2 \ \in \ X, \forall \ t \ \in [0,1] : f(t(x_1) + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2).$$

The objective of mathematical programming is to find the solution x^* that maximizes (or minimizes) a function $f(.)$. A local minimum x^* is defined as the optimal solution if

$$\exists \delta > 0 | \|x - x^*\| \leq \delta \rightarrow f(x^*) \leq f(x) \forall x \in \mathbb{X}$$

where $\mathbb{X}$ is a subset of value.

A convex problem is defined as

$$\min_x f(x) \tag{2.1}$$

$$\text{s.t. } x \in \mathbb{X} \tag{2.2}$$

where $\mathbb{X}$ is a convex set and $f(x)$ is a convex function. $f(x)$ is also known as the objective function of an optimization problem. One characteristic of a convex problem is that every local minimum happens to be a global minimum as well.

A linear programming (LP) model is a subclass of convex optimization which can be formulated as

$$\min_x \ c^T x \tag{2.3}$$

$$\text{s.t. } Ax = b \tag{2.4}$$

$$x \geq 0 \tag{2.5}$$

where $x \in \mathbb{R}^n$ denotes the variables, $c \in \mathbb{R}^n$ denote the coefficient of the objective function, and $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ define the constraints. In a LP model, the objective function (2.3) and the constraints (2.4 and 2.5) are all linear. Interestingly, a LP model can also be interpreted geometrically by a polytope where $Ax = b$ is a set of hyperplanes that limits the feasible region (See a polytope in Figure 2.2 [23]). Moreover, it can be proven that an optimal solution is equal to at least one vertex of the polytope if the LP model is bounded.

The best algorithm found in practice to solve a LP model is the Simplex method. This algorithm takes, in theory, an exponential number of iterations (or exponential time), but it happens to solve most problems in practice in a polynomial number of iterations (or polynomial time). As illustrated in Figure 2.2, the idea of this algorithm is to start from a feasible solution, and to iterate wisely over the vertices of the polytope until it finds a local minimum.

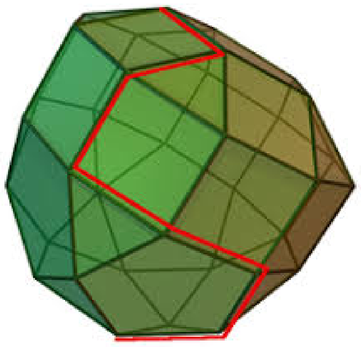


Figure 2.2 Simplex method

2.2.2 Integer linear program

Integer programming (IP) model is a LP model where the variables are constrained to be integers. An IP model can be formulated as

$$\min_x c^T x \tag{2.6}$$

$$\text{s.t. } Ax = b \tag{2.7}$$

$$x \in \mathbb{Z} \tag{2.8}$$

which we denote by P. The integrality constraints makes this class of problem non-convex. Furthermore, some practical problems modeled by an IP model can be solved by exhaustive search, but, in general, no known algorithm has been able to solve IP model in polynomial time making it a member of the NP-Hard family.

Exact methods have been developed to find the optimal solution or near-optimal solution to problems that can be formulated as an IP model. The first step of these exact methods consists of transforming the integer variables into continuous variables. This reformulation denoted as P^* is also known as the LP relaxation of an IP model. The LP relaxation is not unique in general, because it is possible to transform the constraints $(A, b) \rightarrow P^*$ to $(\bar{A}, \bar{b}) \rightarrow \bar{P}$ such that all feasible solutions in P are also in $\bar{P}$ where $P^* \neq \bar{P}$. The ideal reformulation is a polytope named the Convex hull denoted by $Conv(P)$ that includes all feasible solutions in P and where every vertices of $Conv(P)$ is a feasible solution of P (See polygon form by red dot in Figure 2.3 [24]). Unfortunately, no algorithm has been proved to be able to find the $Conv(P)$ in polynomial time. Nonetheless, the cutting plane algorithm was developed to find an approximate $Conv(P)$. The idea is to generate new constraints iteratively such that we remove the feasible solutions of P^* that are not feasible in P to create $\bar{P}$. Geometrically, these new constraints act as a cut of the polytope which explains the name given to this algorithm (See green line in Figure 2.3 defines as a strong cut). There's plenty of different methods to create cuts such as combining constraints, but finding strong cuts remains a difficult task.

Another approach to solve an IP model is the branch and bound algorithm (B&B). The idea of this tree-based method is to iteratively transform the IP model into smaller and easier sub-problems until an optimal solution is found. Every sub-problem is represented as a node in the tree, and a branch is represented by the introduction of a new constraint to a node. For every node, B&B solves the LP relaxation of the IP model, and finds a solution x^*. If x^* satisfies the integer constraints, then this node is defined as a leaf and we can stop branching on it. On the other hand, if x^* does not satisfy the integer constraints, B&B selects a variable x_j that is noninteger, and creates two sub-problems by introducing the constraints $x_j \leq \lfloor x_j^* \rfloor$ and $x_j \geq \lfloor x_j^* \rfloor + 1$, respectively. The optimal solution is found when every node is a leaf or has been pruned. For the specific case where the IP model is bounded and has n binary variables, B&B needs 2^n iterations in the worst-case scenario. Nonetheless, B&B is extremely powerful in practical problems mainly due to the pruning concept. Rather then finding every leaf of a

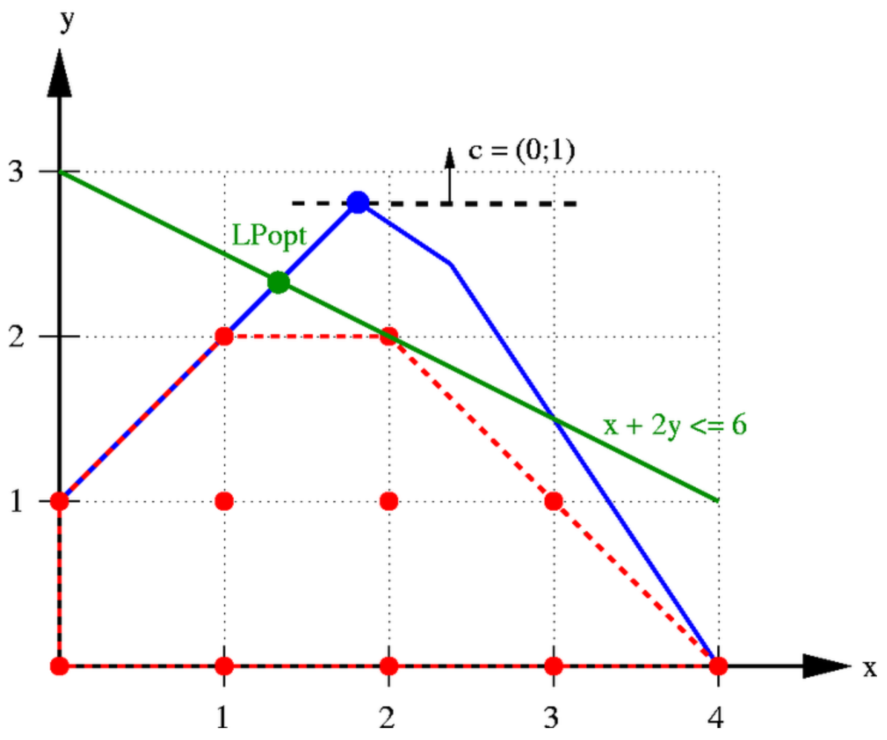


Figure 2.3 Cutting plane algorithm

tree, it is possible to prune sub-trees whenever the LP relaxation of a node is greater than a feasible solution previously found in another node. In other words, assuming we obtain the feasible solution x^* after solving $IP^1 = \{\min c^T x : Ax = b, x \in \mathbb{Z}\}$, we can prune a sub-tree whenever the solution $\bar{x}$ of its LP relaxation gives $c^T x^* \leq c^T \bar{x}$. Enabling pruning of a tree is explained by the fact that the solution to the LP relaxation represents a lower bound on the possible solution found by the sub-tree, meaning that no solution in this sub-problem can be better than x^*.

The branch and bound algorithm and cutting plane algorithm combine with the LP relaxation form the basis of every IP solver. Note that this formulation is generalized by the Mixed Integer Linear Program (MILP) which deals with continuous and integer variables, but the idea to solve this class of problem remains the same as what we detailed for the IP.

Combinatorial optimization problems can often be modeled as an IP, but some of them remain extremely difficult to solve even with commercial solvers. Fortunately, we sometimes have a good understanding of a DO problem enabling us to build a simulation model that can produce a feasible solution. Local search algorithms is a class of heuristics that tries to leverage the simulation model to find the optimal solution. These algorithms use an objective function, also known as the fitness function of an heuristic, to evaluate the quality of a solution. Heuristics do not guarantee convergence to global optimality, but they are able in practice to find good solutions in a reasonable amount of time. Genetics algorithm (GA) is one heuristic among this family that is based on the idea of natural selection. While most heuristics perform greedy operations on a single solution, GA rather focus on generating a population of solutions (or individuals) that evolves from generation to generation. The typical step of a GA goes as follows:

1. Initialize first generation with N individuals $S_1^0, ..., S_N^0$.
2. Evaluate the quality of S_i^j $\forall i \in 1, ..., N$ with the fitness function.
3. If all individuals are identical or maximum time is reached, stop the GA.
4. Otherwise,
    - Select pairs of individuals based on their fitness.
    - Mate the pairs of individuals to create a new generation with N individuals $S_1^{j+1}, ..., S_N^{j+1}$.
    - Execute mutation on $\alpha\%$ of the new individuals to include feasible solutions that were not in the initial set.
    - Restart step 2 with the generation $j + 1$.

2.3 Machine learning

Machine learning (ML) has gained popularity in the last decade with important breakthroughs in computer vision (e.g. [25]) and in reinforcement learning (e.g. [26], [27]). ML includes any algorithm and statistical model that leverages the information gathered from past experience to develop models that have the ability to predict outcomes in the future.

There are three common paradigms in ML: (1) Supervised learning, (2) Unsupervised learning, and (3) Reinforcement learning. While unsupervised learning tries to find pattern in unlabeled data, and reinforcement learning seeks to find the optimal solution by interacting with an environment, this thesis rather focuses on supervised learning, which consists of developing a predictive model out of collected labeled data. This section will introduce the ML concept used throughout the rest of this thesis. We refer the reader to the book *The elements of statistical learning* by Hastie et. al [28] and the book *Deep Learning* by Goodfellow et al. [29] for more details on ML.

2.3.1 Supervised learning

In supervised learning, one tries to estimate an output (or a collection of outputs) using historical data. A dataset is divided into two parts: (1) the independent variables often denoted as the inputs or the features of the problem, and (2) the dependent variable also known as the target variable, the outcome or the output of a problem. While convex optimization tries to find the variables (i.e., x in a LP) that optimize an objective function, a ML algorithm, in particular linear regression, rather focuses on finding the parameters of the objective function (i.e., the coefficient vector c in a LP model) that approximates best the output. ML algorithms use the features from a dataset to train a model to predict the target variable. The target variable can either be continuous value or discrete set of classes. A model predicting a continuous output such as a stock price performs a regression task. On the other hand, a model predicting a discrete target variable such as the expected number of claims in the next year by an insured individual or whether or not a person will win a sports bet is described as a classification task.

Developing a ML model includes three phases: (1) a training phase, (2) a validation phase, and (3) a testing phase. By using a dataset also known as the training set, the training phase consist of learning rules or fitting a distribution able to infer the outcomes of the training set. The validation phase then evaluates the performance of the trained model over a validation set, and aims at tuning our model such that it generalizes best to unseen data. Finally, the testing phase uses an independent dataset defined as the testing set to evaluate the accuracy of the model.

Loss function

An important component of a ML model is the *loss function*, also known as the *cost function*. The loss function is chosen according to the type of problem we want to solve. The loss function penalizes the error made by the predictive model. Therefore, a common goal of a ML algorithm is to minimize the value of the loss function, which relates to minimizing the errors made by the predictive model, by choosing parameters. Mathematically, one can write this as

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{m} L(y^{(i)}, x^{(i)}; \theta)$$

where θ are the parameters of the problem and $L(\cdot)$ is the loss function utilized.

Maximum likelihood estimation (MLE) is a common approach used for identifying optimal parameters of a probability distribution for fitting to data. MLE uses the likelihood function of a probability model, and uses gradient of the likelihood function to find the optimal value for each parameter. MLE can be interpreted as the estimator that minimizes the difference between the observation and the prediction. Two examples of popular loss functions are the mean squared error (MSE) and the cross-entropy loss.

The former can be derived using the MLE of the normal distribution, and is calculated as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\widehat{f(x_i;\theta)} - y_i)^2$$

where N is the size of the training set, $\widehat{f(x_i;\theta)}$ is the predicted value given the features x_i and the parameters θ, and y_i is the true value associated to x_i. As the name implies, MSE calculates the average squared difference between the predicted value and its true value, and it is a common *loss function* use in regression task. A $MSE = 0$ indicates a perfect prediction. The cross-entropy loss is often use as *loss function* for classification task, and it measures the difference between the true probability distribution and the predicted probability distribution. The problem with this *loss function* is that it aims at decreasing the classification error. Let's assume we have to predict the probability that team A wins against team B, and team A won 99 of the 100 last games played between both teams. By predicting team A with a probability of 1 over B will ends up having an almost perfect score of cross-entropy loss. In order to have a better understanding of such a predictive model, one could decide to evaluate its performance by using the Receiver Operating Characteristic (ROC). This metric depicts the difference between the true positive rate and the false positive rate, which corresponds to the number of samples that are correctly classified as positive to the number of samples that are misclassified as positive, respectively. A perfect ROC has an area under the curve (AUC) equal to 1.

Generalization

ML aims at building a model that can generalize to make predictions on unseen data. The capacity of a model is defined by the ability to find complex structures in the data. The capacity of a model depends on the number of hyperparameters. Models with high capacity tend to memorize or overfit the outcome of the training set, leading to a highly accurate model in the training phase with poor performance on the testing set. On the other hand, models with low capacity tend to underfit because they are unable to find structure in the data that helps predicting the outcome. As illustrated in Figure 2.4, the optimal fit is the one where the average error on the validation set restarts to increase while the one on the training set continue to decrease.

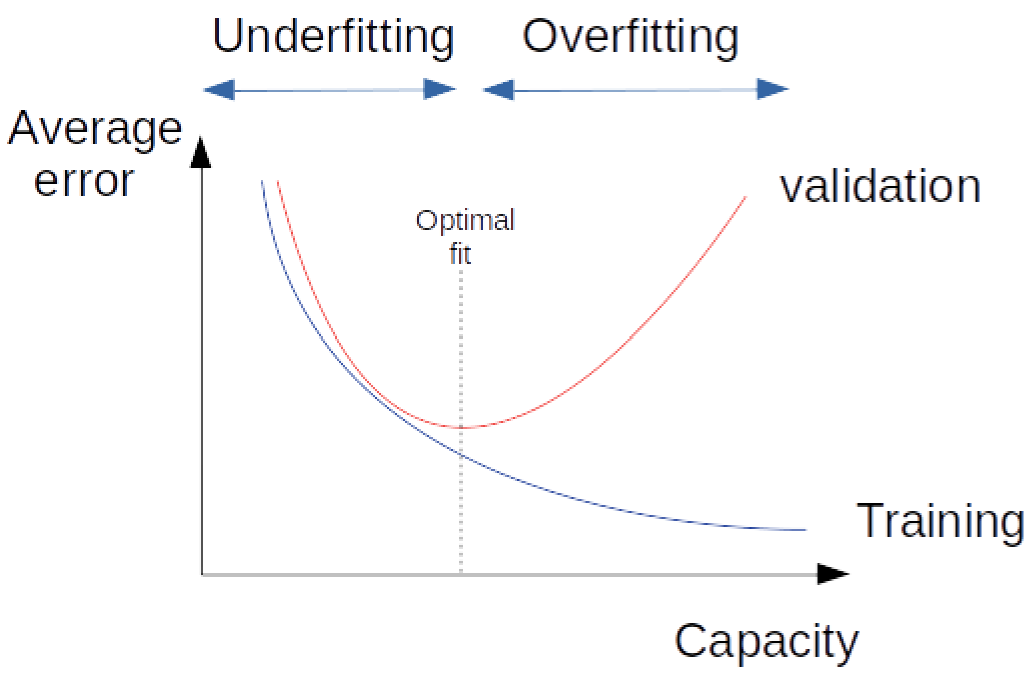


Figure 2.4 Overfitting vs underfitting

Regularization is a technique that controls the training process in a way that limits overfitting on the training set. One way is to add a penalty term to the loss function in order to reduce the impact of outliers on the training process. Another important method that helps control training is cross validation. Cross validation leverages the validation phase introduced earlier to select hyperparameters that generalize best to unseen data. This algorithm also helps in situations where we do not hold out much data. See Algorithm 1 for more details.

Algorithm 1: K-fold cross-validation algorithm [29]

```
Required: The dataset $\mathcal{D}$ with label $y$.
Required: The ML algorithm $f$ that takes a dataset as input and a set of
  hyperparameters. $f$ outputs a learned function.
Required: The loss function $L$ that takes a vector of predicted value and the true
  labels $y$. $L$ returns a real value.
Required: The number of folds $k$
Required: Initialize $\xi$ as a set of random combination of hyperparameters indexed by
  $\xi_i \forall i = 1, \ldots, k$
Split $\mathcal{D}$ in $k$ mutually exclusive subsets denoted as $\mathcal{D}_i$ with label $y^{(i)}$. Their union is
  $\mathcal{D}$
$i = 0$
$min_e = +\infty$
$f_{min} = f(\mathcal{D} \setminus \mathcal{D}_i; \xi_i)$
while $i \leq k$ do
    $z_i = f(\mathcal{D} \setminus \mathcal{D}_i; \alpha_i)$
    $e_i = L(z_i(\mathcal{D}_i), y^{(i)})$
    if $e_i \leq min_e$ then
        $min_e = e_i$
        $f_{min} = z_i$
    end if
    $i+ = 1$
end
return $min_e$, $f_{min}$
```

Logistic regression

A logistic regression is a probability model, also known as a binary classifier. It takes as input a set of features and outputs a number between 0 and 1. The function it learns is

$$P(y = 1|X = x) = \frac{1}{1 + e^{-\theta^T x}}$$

where θ represents the parameters of the logistic regression. A logistic regression is a member of the *Generalized Linear Models* family. It is sometimes considered a linear model due to the linear association between θ and X. The log-likelihood of the logistic regression is

$$logloss = \sum_{i=1}^{N} -(y_i \log(P(y_i = 1|X = x_i)) + (1 - y_i) \log(1 - P(y_i = 1|X = x_i))),$$

and this loss is also known as the binary cross-entropy. To regularize this model, we can add a penalty term to the loss function. It becomes

$$logLoss = \sum_{i=1}^{N} -(y_i \log(P(y_i = 1|X = x_i)) + (1 - y_i) \log(1 - P(y_i = 1|X = x_i))) + \frac{\lambda}{2}\|\theta\|$$

. The penalty term $\frac{\lambda}{2}\|\theta\|$, also known as the weight decay, increases the capacity of the model by adding the hyperparameter λ. It makes sure that the value of the parameters of the model do not increase too much. The best value of λ can be established by using the K-fold cross-validation algorithm.

The gradient descent algorithm is a common way to optimize a ML algorithm. This algorithm updates θ iteratively using the gradient of the loss function until a stopping criteria is met. This algorithm summarizes to

$$\theta^{t+1} = \theta^t - \alpha \nabla_\theta \sum_{i=1}^{N} L(y^{(i)}, x^{(i);\theta}) \tag{2.9}$$

where α is the learning rate and ∇_θ is the gradient of the loss function with respect to θ. For the logistic regression, it can be proven that the updates rule for the logistic regression is

$$\theta = \theta - \alpha \nabla_\theta \sum_{i=1}^{N} \left(\frac{1}{1 + e^{\theta^T x^{(i)}}} - y \right) x^{(i)}. \tag{2.10}$$

Neural network

Neural networks (NN), also known as multilayer perceptrons, are a generalized form of several classical regression/prediction models. As we can see in Figure 2.5, a NN is characterized by a network structure that connects many functions. A NN is divided into three sections: (1) the first layer known as the input layer, (2) the last layer known as the output layer, and (3) the layers in between defined as the hidden layers.

Each arrow in Figure 2.5 defines the weights or the parameters of the model while each circle in the hidden layer defines the neurons characterized by an activation function. NN linearly combines inputs together, and transforms their weighted combination by passing them through an activation function. NNs are extremely flexible models that can adapt

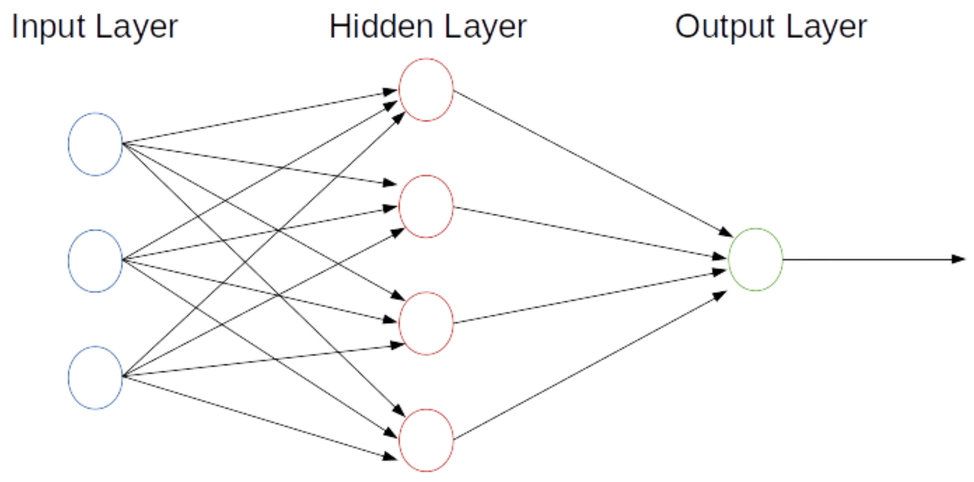


Figure 2.5 Neural network structure

to representing very general functions. In fact, the universal approximation theorem says that a NN with one hidden layer can approximate any continuous function to an arbitrary precision. NNs can perform classification tasks as well as regression tasks by choosing the right activation function and the number of units needed for a given task in the output layer. Moreover, the learning capacity of a NN increases as we increase the number of layers or the number of neurons per layer.

Mathematically, a feedforward NN is represented by the following equations

$$h^{(1)} = g^{(1)}\left(\mathbf{W^{(1)T}} h^{(0)}\right) \tag{2.11}$$

$$\vdots \tag{2.12}$$

$$h^{(i-1)} = g^{(i-1)}\left(\mathbf{W^{(i-1)T}} h^{(i-2)}\right) \tag{2.13}$$

$$h^{(i)} = g^{(i)}\left(\mathbf{W^{(i)T}} h^{(i-1)}\right) \tag{2.14}$$

where $h^{(z)}$ designs the vector of outputs of the zth layer with h^0 and h^i equal to the input and the output, respectively. $\mathbf{W^{(z)T}}$ is the transpose of the matrix of parameters and g^z is the activation function of layer z. It is known as the forward propagation of the NN.

Back-propagation is an algorithm used to train a NN. Given the structure of the NN, the algorithm finds $\mathbf{W}$ that minimizes a chosen loss function. Due to the activation function of a NN which are mostly nonlinear function, back-propagation iteratively uses the gradient descent algorithm to minimize the loss function. Since a NN is structured in a chain, the back-propagation algorithm uses the chain rule of calculus to propagate the gradient to all the parameters in the NN. The reader is referred to Goodfellow [29] for more details on the back-propagation algorithm.

To improve the training of a NN, one can utilize *stochastic gradient descent* (SGD) with momentum. SGD has the particularity to train the model using the average gradient of a minibatch, a sample without replacement of h examples of the training set, rather than using the gradient of the loss function evaluated on the entire dataset. Minibatch tends to be a source of noise for which the gradient does not vanish. Once the gradient has been computed over the entire training set, a new epoch starts which simply continue the training process by resampling without replacement new minibatchs from the same training set. Moreover, it is possible to add momentum to SGD to accelerate the training of a NN. Momentum adds a parameter to SGD that accumulates the gradient. The idea behind this modification is that the speed and direction from previous gradients should also impact the next update of the parameters. Therefore, if the direction between two consecutive gradients remains the same, the momentum parameter will accelerate the learning by updating the parameters from a factor of the two last gradients. On the other hand, if two consecutive gradients move in two directions, the momentum parameter will update the parameter without forgetting the update that was previously made.

Algorithm 2: Stochastic Gradient Descent with Momentum [29]

```
Required: Learning rate $\alpha$, momentum parameter $\omega$
Required: Initialize $\theta$ and velocity $\nu$
while stopping criterion not met do
    Sample a minibatch of m exemple from the training set $\{x^{(1)}, \dots, x^{(m)}\}$ with
      corresponding $y^{(i)}$,
    Compute the gradient estimate $\nu \leftarrow \omega\nu - \alpha\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta), y^{(i)})$.
    Update: $\theta \leftarrow \theta + \nu$.
end
```

Finally, NN can have a high capacity. To control the risk of overfit, we use early stopping, which is a stopping criteria that forces the NN to stop training once the error made on the validation set increases for N consecutive epochs.

2.4 Embedding NNs in optimization (`JANOS`)

There is a close relationship between optimization and ML. As discussed in the previous section, ML algorithms search for parameters (and hyperparameters) that minimize the error of a loss function which itself is an optimization problem. Recent research has focused on the opposite direction, namely using ML to improve optimization [30]. Some work focused on using reinforcement learning to decide which heuristic to use to solve a combinatorial problem (e.g., [31]) while other work improved the branch-and-bound algorithm using a deep learning model (e.g., [32]).

`JANOS` [33] is a solver that works at the frontier of ML and optimization. Using a predictive-prescriptive framework, `JANOS` aims at optimizing a decision-making problem given a pre-trained predictive model using the python package Scikit-Learn[34]. `JANOS`'s solves problems that can be modeled as the following MILP:

$$\max_{x} \sum_{j=1}^{n_1} c_j x_j + \sum_{k=1}^{n_2} d_k y_k \tag{2.15}$$

$$\begin{aligned}
\text{s.t.} \quad & \sum_{j=1}^{n_1} a_j^i x_j \le b_i, && \forall i \in \{1, \dots, m\} && (2.16)\\
& y_k = g_k(\alpha_1^k, \dots, \alpha_{pk}^k; \theta_k), && \forall k \in \{1, \dots, n_2\} && (2.17)\\
& \alpha_l^k = e_l^k \cdot x, && \forall l \in \{(q_k + 1), \dots, p_k\}, k \in \{1, \dots, n_2\} && (2.18)\\
& x_j \in X_j && \forall j \in \{1, \dots, n_1\}. && (2.19)
\end{aligned}$$

x_j defines the *regular variable* of the MILP and a_j^i, b_i, c_j and d_k are the parameters. Moreover, y_k are the *predicted variables*, which represents the outcome of the predictive model g_k with the features α_l^k and the parameters θ_k, and e_l^k is a binary unit vector of length n_1 with 0 everywhere except at the coordinate of the associated *regular variable.* As of now, `JANOS` allows the user to use three different predictive models: (1) linear model, (2) logistic regression, and (3) NN with rectified linear activation function (ReLU) as activation function.

2.4.1 NN with Rectified Linear Unit Activation Function in Janos

A ReLU activation function is defined as

$$g^i\left(\mathbf{W^{(i)T}} h^{(i-1)}\right) = \max\left(\mathbf{W^{(i)T}} h^{(i-1)}, 0\right).$$

This nonlinear activation function is linear on half of the domain, and it is equal to 0 on the other half. Training a NN with such nonlinear activation functions allows the modelling of a complex linking between input variables. A main advantage of this activation function

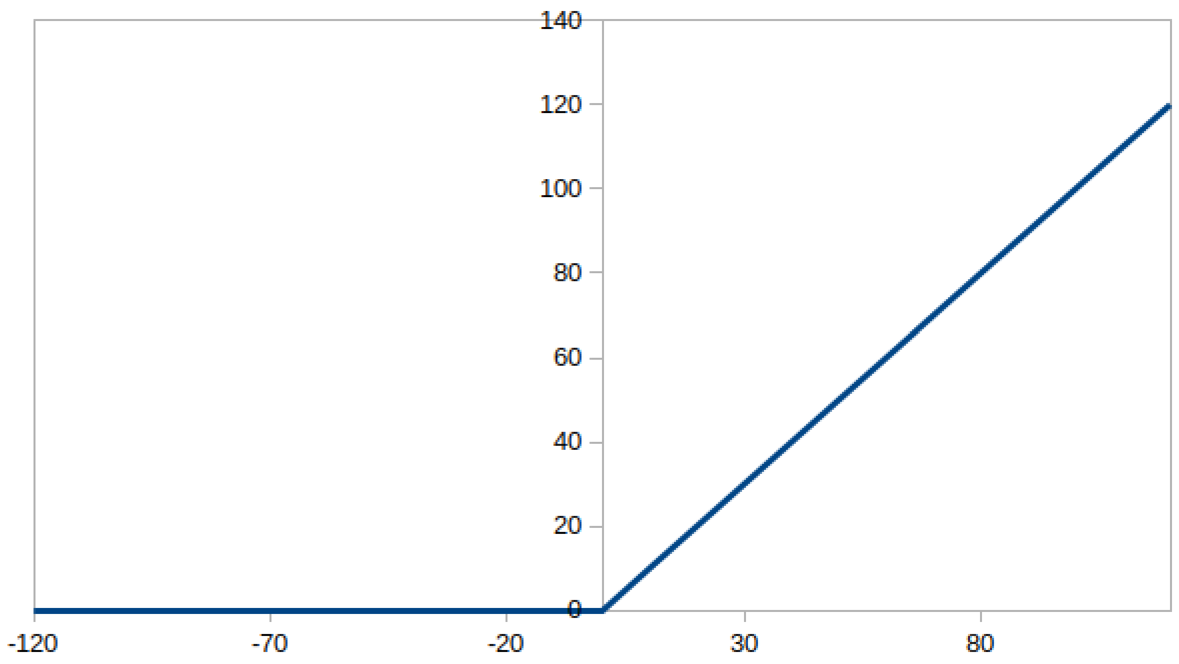


Figure 2.6 ReLU activation function

compared to other commonly used activation functions is that the gradient of the linear part of a ReLU remains large making it easier to optimize.

An interesting aspect of a NN with ReLU activation functions is that it can be represented with a network flow formulation [35]. The NN can be seen as an acyclic layered diagraph $N = (V, A)$ where V is the collection of all layers $V = \{V_1, \dots, V_l\}$, and A is the collection of all arcs. All arcs $a = (u, v)$ in A are directed arcs from node u in layer V_j to node v in layer V_{j+1}. $w(a)$ will denote the weight associated to arc a and $B(u)$ will denote the learned bias associated to node u. Given the weights of the NN and the vector of inputs which are respectively θ_k and α^k in (2.17), `JANOS` replaces the nonlinear predictive model by reformulating the NN as

$$
\begin{aligned}
y_k &= F_t && && (2.20)\\
F_v &= G_v, && \forall v \in V_1 \cup V_l && (2.21)\\
G_v &= \alpha(v), && \forall v \in V_1 && (2.22)\\
G_v &= \sum_{u \in V_{j-1}} w((u,v)) \cdot F_u + B(u), && \forall j \in \{2,\ldots,l\}, v \in V_j && (2.23)\\
-M \cdot (1 - z_v) \leq G_v &\leq M \cdot z_v, && \forall v \in V_2 \cup \ldots \cup V_{l-1} && (2.24)\\
G_v - M(1 - z_v) \leq F_v &\leq G_v + M \cdot (1 - z_v), && \forall v \in V_2 \cup \ldots \cup V_{l-1} && (2.25)\\
0 \leq F_v &\leq M \cdot z_v, && \forall v \in V_2 \cup \ldots \cup V_{l-1} && (2.26)\\
z_v &\in \{0,1\}, && \forall v \in V && (2.27)\\
G_v, F_v &unconstrained, && \forall v \in V. && (2.28)
\end{aligned}
$$

y_k is the output of the NN, F_v the linear transformation done by each layer, G_v is the value obtained after applying the ReLU function on F_v and z_v is a binary variable required to linearize the ReLU function. The constraints (2.20), (2.21), (2.22) and (2.23) represents the forward propagation of the NN while the big-M linear constraints (2.24), (2.25) and (2.26) represents the ReLU activation function. We invite the reader to read Bergman et al. [33] for more details on this reformulation. `JANOS` allows the user to use `Gurobi` to find an optimal solution, or a near-optimal solution, to a MILP model with an embedded predictive model.

2.5 Optimizing the expected value of order statistics

Bergman et al. [36] is the first paper to discuss the complexity of a decision-making problem that requires the optimization of the expected score of order statistics. Mathematically, in that paper, the authors study problems of the form

$$\max_{x \in \Omega \subseteq \{0,1\}^n} E(Y_{(k)}(x)) \tag{2.29}$$

where $E(\cdot)$ is the expected value of $Y_{(k)}$ which is the k^{th} order statistic. The k^{th} order statistic is the smallest k^{th} value of the variables, with the largest order statistics being the maximum among a collection of random variables, similar to what we study in this paper. The authors shows that this problem classifies as a NP-hard problem, and they introduce an exact algorithm to solve it in certain cases.

2.6 Optimization for *March Madness*

In order to address the problem of selecting an entry for a *March Madness* betting pool, Kaplan et al. [37] introduced a *dynamic programming* approach that identifies a bracket that maximizes the expected number of points ($\max_{b \in \mathcal{B}}(E(S(b)))$ among all possible brackets. The algorithm requires as input the probability that every team will reach their game in every round, denoted as $Q_{t,r}$, which is calculated as follows:

Algorithm 3: Recursive calculation of $Q_{t,r}$

```
Let $o(t,r)$ be a list of all possible opponents of team $t$ in round $r$;
Initialize $Q_{t,1} = P_{t,o(t,1)}$ and $r = 2$;
while $r < |R|$ do
    $Q_{t,r} = Q_{t,r-1} \cdot \sum_{t' \in T(G(t,r))/t} P_{t,t'} \cdot Q_{t',r-1} \; \forall \, t \; \in T.$
end
```

Following the calculation of these probabilities, the algorithm then calculates the expected number of points of any pick in any round, denoted as $\mu_{t,r}$, evaluated through the following recursive algorithm:

Algorithm 4: Recursive calculation of $\mu_{t,r}$

```
Let o(t,1) be the opponent of team $t$ in round 1;
Initialize $Q_{t,1} = P_{t,o(t,r)}$, $r = 2$ and $\mu_{t,1} = 2^{1-1} \cdot Q_{t,1}$;
while $r < |R|$ do
    $\mu_{t,r} = \mu_{t,r-1} + \max_{t' \in o(t,r)} \mu_{t',r-1} + Q_{t,r} \cdot 2^{r-1} \; \forall \, t \; \in T$
    $r = r + 1.$
end
```

Using $\mu_{t,r} \forall r \in R, t \in T$, the authors then use a greedy algorithm that selects the teams with the maximum expected number of points for every games by starting from the final game and moving backwards through the tournament.

This dynamic programming approach can easily be transformed into an IP model which we denote as $\texttt{IP}^{\texttt{MM}}$. Using the same probabilities $Q_{t,r}$ as in Kaplan et al. [37], an IP model can be written as

$$\max_{x} \sum_{g\in G} \sum_{t\in T(g)} 2^{R(g)-1} Q_{t,R(g)} x_{i,R(g)} \tag{2.30}$$

$$\begin{aligned} \text{s.t.} \qquad & x_{t,r} \leq x_{t,r-1}, & \forall t \in T, r \in R \qquad (2.31) \\ & \sum_{t\in T(g)} x_{t,R(g)} = 1, & \forall g \in G \qquad (2.32) \end{aligned}$$

where

$$x_{t,r} = \begin{cases} 1 \text{ if team } t \text{ wins in round } r, \\ 0 \text{ otherwise.} \end{cases}$$

The objective function calculates the expected number of points of a bracket. The constraints ensure that every team that wins in round r must have won in every previous round and that only one team can win a game. We define these two constraints as *bracket feasibility constraints.*

We have now described two algorithms that find a single entry with the maximum expected score in a *March Madness* betting pool. The first is the dynamic program introduced by Kaplan et al. [37], and the second is our IP model $\texttt{IP}^{\texttt{MM}}$.

In this thesis we study the problem of finding k brackets that maximizes the expected score of the maximum scoring bracket. This problem is very challenging because there exist $\binom{2^{63}}{k}$ ways of selecting k entries. This problem can be expressed as

$$\max_{b_1,\ldots,b_k\in\mathcal{B}} E\left(\max_{b_j\in\{b_1,\ldots,b_k\}} S(b_j)\right).$$

This formulation expands to

$$\max_{b_1,\ldots,b_k\in\mathcal{B}} \sum_{b^*\in\mathcal{B}} P(b^*) \left(\max_{b_j\in\{b_1,\ldots,b_k\}} S\left(b_j, b^*\right)\right).$$

Assuming we want to solve this problem for $k = 2$, where the entries are denoted by b_1 and b_2, we can rewrite the previous formulation as

$$\max_{X} \sum_{b^*\in\mathcal{B}} P(b^*) \left(\max\left(\sum_{g\in G} 2^{R(g)-1} x_{b^*[g],R(g),1}, \sum_{g\in G} 2^{R(g)-1} x_{b^*[g],R(g),2}\right)\right) \tag{2.33}$$

where

$$x_{t,r,k} = \begin{cases} 1, \text{ if we select team } t \text{ in round } r \text{ for the entry } k, \\ 0, \text{ otherwise.} \end{cases}$$

Equipped with variables $x_{t,g,k}$, we add the *bracket feasibility constraints* to compose a baseline IP model for finding the optimal two brackets:

$$\max_X \sum_{b^* \in \mathcal{B}} P(b^*) \left(\max \left(\sum_{g \in G} 2^{R(g)-1} x_{b^*[g],R(g),1}, \sum_{g \in G} 2^{R(g)-1} x_{b^*[g],R(g),2} \right) \right) \tag{2.34}$$

$$\begin{aligned} \text{s.t.} \quad & x_{t,r,k} \leq x_{i,r-1,k}, && \forall\, t \in T, r \in R \setminus \{1\}, k \in \{1,2\} && (2.35) \\ & \sum_{t \in T(g)} x_{t,R(g),k} = 1, && \forall\, g \in G, k \in \{1,2\} && (2.36) \\ & x_{t,r,k} \in \{0,1\}, && \forall t \in T, r \in R, k \in \{1,2\}. && (2.37) \\ & && && (2.38) \end{aligned}$$

This IP model remains challenging to solve due to the max$(\cdot)$ function, which is nonlinear, and it has exponential size. Fortunately, it is possible to linearize the max$(\cdot)$ function. The new formulation goes as

$$\max_{x,h,s,z} \sum_{b \in \mathcal{B}} P(b) h_b \tag{2.39}$$

$$
\begin{aligned}
\text{s.t.} \quad & x_{t,r,k} \leq x_{t,r-1,k} && \forall\, t \in T, r \in R \setminus \{1\}, k \in \{1,2\} && (2.40)\\
& \sum_{t \in T(g)} x_{t,R(g),k} = 1 && \forall\, g \in G, k \in \{1,2\} && (2.41)\\
& \sum_{g \in G} 2^{R(g)-1} x_{b[g],R(g),k} = s_b^k && \forall\, b \in \mathcal{B}, k \in \{1,2\} && (2.42)\\
& s_b^1 - Mz[b] \leq s_b^2 && \forall\, b \in \mathcal{B} && (2.43)\\
& s_b^1 \geq s_b^2 - M(1 - z_b) && \forall\, b \in \mathcal{B} && (2.44)\\
& h_b \leq s_b^1 + M(1 - z_b) && \forall\, b \in \mathcal{B} && (2.45)\\
& h_b \leq s_b^2 + Mz_b && \forall\, b \in \mathcal{B} && (2.46)\\
& x_{t,r,k} \in \{0,1\} && \forall t \in T, r \in R, k \in \{1,2\}, b \in \mathcal{B}. && (2.47)
\end{aligned}
$$

where s_b^k is the score of entry k when the outcome bracket b is observed and h_b is the maximum score of both entries when the outcome bracket b is observed. (2.43), (2.44), (2.45) and (2.46) are all Big-M constraints that combine together linearize the $\max(\cdot)$ function of the previous formulation. This IP model is very difficult to solve due the fact that the number of constraints and variables depends on $\mathcal{B}$, which contains 2^{63} brackets. We explored the idea of solving this model assuming that we replace $\mathcal{B}$ by a sample of N brackets. However, this model contains $|T| \cdot |R| \cdot 2 + 3 \cdot N$ variables and $2 \cdot |G| + |T| \cdot (|R| - 1) \cdot 2 + 6 \cdot N$ constraints which is still very difficult to solve. So the question remains: how do we find the collection of k brackets that maximize the expected value of the maximum scoring bracket? We observe two main challenges in finding the optimal solution of the maximum of k entries: (1) how do we evaluate $E\left(\max_{b_i \in \{b_1,\ldots,b_k\}} S(b_i)\right)$, and (2) how do we find the optimal k entries among 2^{63} feasible brackets? The remainder of the dissertation focuses on answering these questions.

CHAPTER 3 METHODOLOGY

3.0.1 Exact approach

A first approach to evaluate exactly the objective function for k brackets is a tree-based algorithm. Given k brackets, we break the tournament outcome into scenarios that have an impact on the number of points that can be achieved by the k brackets. This method can be seen as building a tree, where each branch conditions on the result of a game and each node contains the minimum and the maximum number of points that each of the brackets can achieve in the scenario defined by the branches that define the node.

Figure 3.1 depicts how the tree-based algorithm finds the expected score of the maximum scoring entry of the four-team tournament presented earlier with b_1 and b_6 as the selected entries.

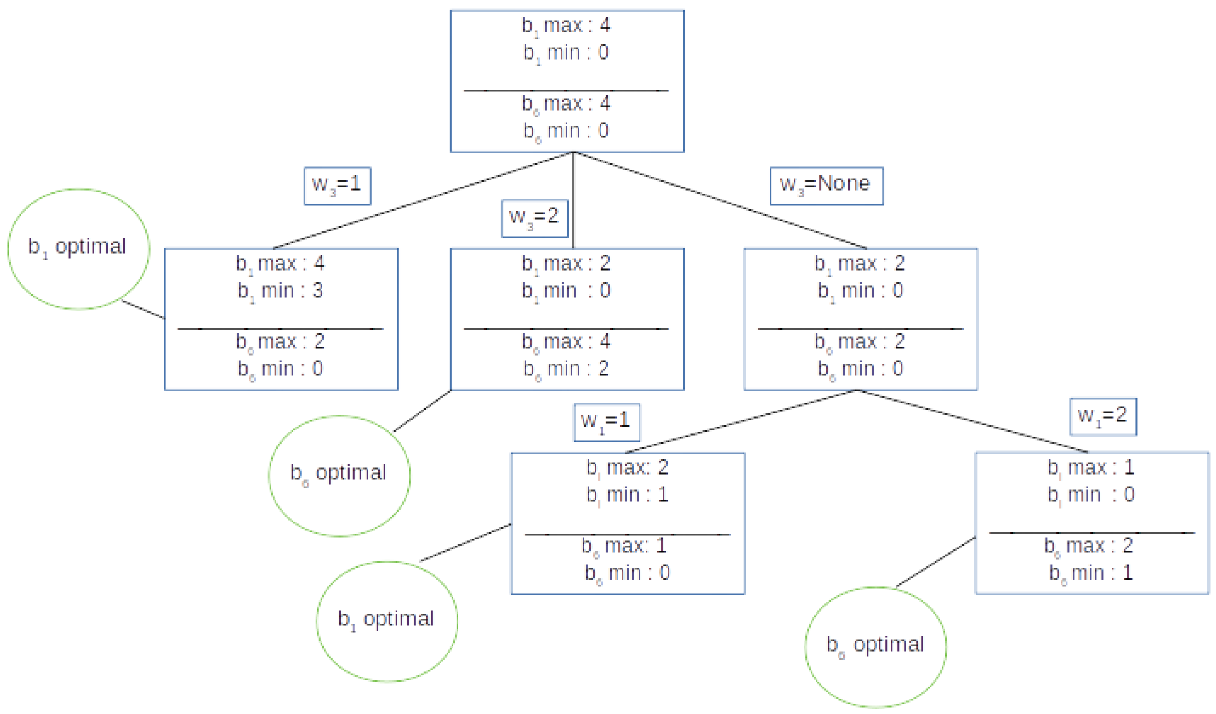


Figure 3.1 Tree-based algorithm example

The order of branching starts with the final game and finishes with first round matches, when required. Note that we always branch on the scenario with the highest probability of occurring. We keep branching until $2^r - 1$ games are reached or if the minimum number of points of any of the selected brackets is greater than the maximum number of points of all other brackets at any point in the tree. This stopping criteria ensures that one bracket will score at least as high as all other brackets, independent of the outcome of the remaining, unconditioned games.

Based on Figure 3.1, we can observe that b_1 is the best entry in scenario 1 and 3, and b_6 is the best entry in scenario 2 and 4. For each node, there are a maximum $(k+1)$ ways of conditioning. In other words, for a given game, the outcome can be any pick made by one of k brackets or none of these picks. An interesting aspect of this branching structure is that conditioning on the winner of a game also gives us the winner of every match where this pick previously played in the tournament. See Table 3.1 to observe the number of points associated with correctly picking or the number of points lost for a wrong selection in any round in a 64-team tournament.

Table 3.1 Tree-based algorithm branching rules

	Increase minimum number of points	Reduce maximum number of points
First round	+1	-1
Second round	+3	-2
Third round	+7	-4
Fourth round	+15	-8
Fifth round	+31	-16
Sixth round	+63	-32

For the two-entry model in a 64-team tournament, there is a maximum of 3^{63} nodes that can be created. The maximum number of points of the 64-team *March Madness* tournament is 192 and the minimum is 0.

Given the tree, we can then compute the probability that a scenario occurs. The probability and the score allows us to evaluate exactly $E\left(\max_{b_j\in\{b_1,\ldots,b_k\}} S(b_j)\right)$. Using the tree of Figure 3.1, the expected score calculates as

$$\begin{aligned} E\left(\max_{b_j\in\{b_1,b_2\}} S(b_j)\right) &= P_{\mathcal{B}_{w_3=1}} \cdot S'(b_1,\mathcal{B}_{w_3=1}) + P_{\mathcal{B}_{w_3=2}} \cdot S'(b_6,\mathcal{B}_{w_3=2}) + \\ &\quad P_{\mathcal{B}_{w_3=\text{None},w_1=1}} \cdot S'(b_1,\mathcal{B}_{w_3=\text{None},w_1=1}) + P_{\mathcal{B}_{w_3=\text{None},w_1=2}} \cdot S'(b_6,\mathcal{B}_{w_3=\text{None},w_1=2}) \\ &= P_{1,2}\cdot(P_{3,4}\cdot P_{1,3}\cdot 4 + P_{4,3}\cdot P_{1,4}\cdot 3) + P_{2,1}\cdot(P_{4,3}\cdot P_{2,4}\cdot 4 + P_{3,4}\cdot P_{2,3}\cdot 3) + \\ &\quad P_{1,2}\cdot(P_{3,4}\cdot P_{3,1}\cdot 2 + P_{4,3}\cdot P_{4,1}\cdot 1) + P_{2,1}\cdot(P_{3,4}\cdot P_{3,2}\cdot 1 + P_{4,3}\cdot P_{4,2}\cdot 2) \end{aligned} \tag{3.1}$$

which is equivalent to equation (1.1).

An interesting aspect of this methodology is that we can obtain upper and lower bounds of the expected score given a limited amount of computational time and/or memory. To do so, the algorithm can simply stop at any point. For every node where the stopping criteria is reached, we compute the expected number of points associated with this node. All the remaining nodes have bounds that intersect between at least one pair of brackets, preventing us from defining the best entry in the scenario defined by the branching rules of the node. In this case, to define the lower and upper bounds of the expected score of the maximum of k entries on a scenario, we assume the lower bound to be the highest minimum score in the node and the upper bound to be the highest maximum score in the node.

For example, suppose a scenario where bracket A has a minimum score of 100 and a maximum score of 160, and bracket B has a minimum score of 120 and a maximum score of 144. The lower and upper bound on the expected score of this scenario for these two brackets is 120 points and 160 points, respectively. To determine the lower bound of the expected score of the maximum of two entries, we sum the lower bound of each scenario, scaled by the probability. The same process is conducted to find the upper bound.

3.0.2 Simulation approach

Given a probability matrix P that estimates the probability that team t_i beats team t_j, we can simulate the outcome of an entire tournament. More specifically, given an initial tournament with |G| teams, these teams are separated in $2^{|R|-1}$ games. We can use P to simulate the outcome for each of these games, and the simulated winners then advance to the next round. The winners of the first round matches are then split in $2^{|R|-2}$ games where we again use P to simulate the winner of the second round matches. By repeating this process for $|R|$ rounds, we simulate a complete bracket (See Figure 1.1 for a visualization of the structure of the tournament).

A second approach to calculating the expectation of a k-bracket solution is *sample average approximation* (SAA). SAA leverage the simulation model to evaluate the expected score of the maximum of k entries. The following provides pseudo-code to the SAA algorithm.

```
Algorithm 5: Sample average approximation of $E\left(\max_{b_i \in \{b_1,\ldots,b_k\}} S(b_i)\right)$

    Required: Simulate $O$ outcome brackets denoted by $b_o^{out}$, $o = 1, \ldots, O$.
    Given a solution of $k$ candidate brackets denoted by $b_c^{can}$, $c = 1, \ldots, k$,
    $E(\widehat{\max_{b_i^{can} \in \{b_1^{can},\ldots,b_k^{can}\}}} S(b_i^{can})) = \frac{1}{O} \sum_{o=1}^{O} \max_{b_i^{can} \in \{b_1^{can},\ldots,b_k^{can}\}} S(b_i^{can}, b_o^{out}))$
```

Note that even though we have an exact approach for the single entry, this approach can also be used for $k = 1$.

3.0.3 Approximation via NNs

Equipped with a simulation model to estimate the expected score of the maximum of multiple entries, we can use it to build a learning-based model. Through simulation, we construct a training set by simulating N random k-brackets solutions, recording their associated expected score. For the input of our NN, we want to leverage the information provided by the structure of the single elimination tournament.

We start by approximating the expected score of a single entry by using a NN with G parameters denoted as f_g. Each of the parameters represents the expected score of the pick selected to win game g, which is denoted as

$$f_g := Q_{b[g],R(g)} \sum_{r=1}^{R} (g) 2^{r-1}. \tag{3.2}$$

This NN is extendable to $k = 2$. To do so, We add the same features for each entry leading to $2 \cdot |G|$ features, and we also include features

$$v_s = I(B_i(s) == B_j(s)) \forall s \in G \tag{3.3}$$

where $I(\cdot)$ is the identity function. (3.3) indicates whether the picks are the same for both brackets. The set of features (3.3) was added because it improved empirically the performance of the NN. Note that these two sets of features can easily be extended to $k > 2$ by adding (3.2) for each of the k brackets and (3.3) for each pair of two brackets for a total of $k \cdot |G| + \binom{k}{2}$ features.

One could decide to learn a different objective function, rather than simply maximizing the expectation of the maximum scoring bracket, depending on the type of strategy we want to exploit. As discussed in Chapter 2, this type of contest often favors solutions with high variance, so one could calculate the variance of all entries using simulation, and train a NN which learns the variance of the score of k entries or a mixture of the expected value and its variance. Another very interesting approach is to learn the expected payoff of an entry. We leave these for future work.

3.1 Optimization algorithms

The previous section introduced three algorithms for calculating $E\left(\max_{b_i \in \{b_1,\dots,b_k\}} S(b_i)\right)$. The next step is to devise an algorithm to identify the optimal collection of brackets among the $\binom{2^{63}}{k}$ possible choices.

3.1.1 Genetic algorithm

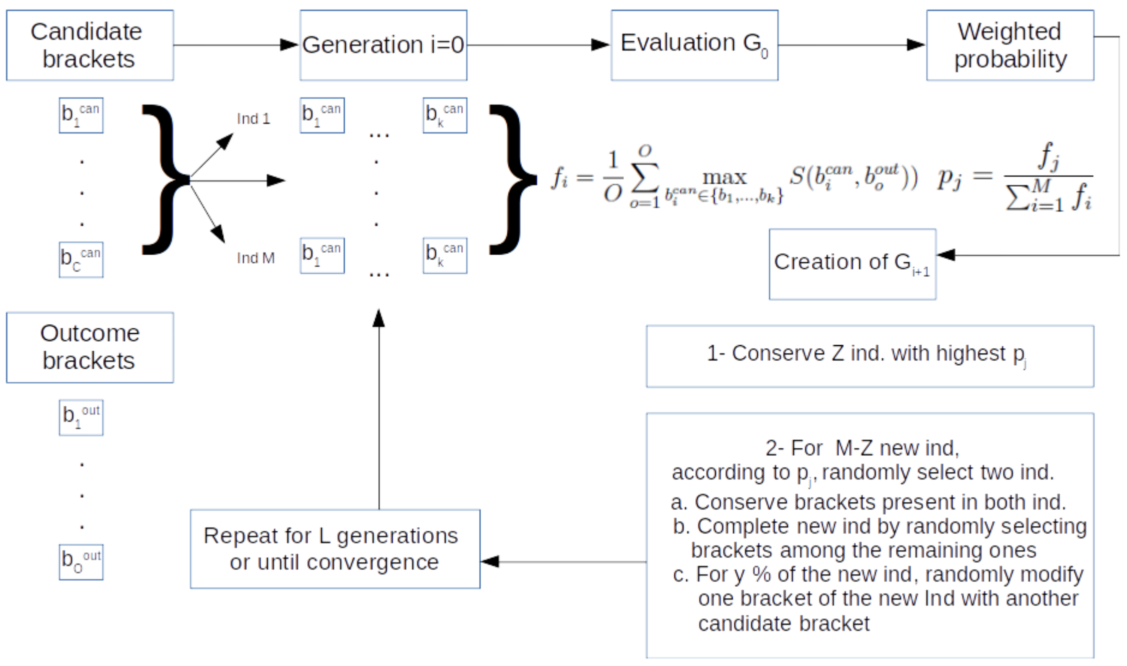
Rather then solving the multi-entry problem through IP, one could randomly try different entries together, observe their empirical performance using simulation, and pick the best among all of them. One such framework utilizes *evolutionary algorithms*, and in particular in this paper we study a *genetic algorithm* (`GA`).

Our `GA` starts by simulating a large collection of C candidate brackets. Each population consists of M individuals defined as $\mathbb{G}_j = \{b_1^{can}, \dots, b_k^{can}\} \forall j \in \{1, \dots, M\}$ where b_i^{can} is one of the candidate brackets. To evaluate the empirical performance of every individual, we use the SAA introduced in Section 3.0.2. We then rank their performance by assigning a normalized average-based probability to each individual. The higher the weighted average of an individual, the higher is the probability for this individual to be selected in a future generation.

To create a new generation of M individuals, we first use elitism, which keeps the Z best individuals from the previous generation. We continue by randomly selecting two individuals from the current generation, and by applying a cross-over operation to create new individuals for the next generation. The cross-over operation mates the two individuals by keeping the intersection and by selecting randomly among the remaining brackets in the two individuals in order to have k brackets in each individual. Finally, we apply the mutation operation which randomly selects an individual from the new generation and modifies one of the k brackets with a randomly generated bracket. These operation are repeated for any number of generations and/or until a time limit is reached. See Figure 3.2 for the details of each step of the `GA`.

3.1.2 NN-Embedded optimization via `JANOS`

We first train a NN, which we denote as $\mathcal{NN}$. As introduced in section 3.0.3, $\mathcal{NN}$ takes as argument the expected score of every selection of an entry. We then adapt $\mathcal{NN}$ to the $\texttt{IP}^{\texttt{MM}}$ model (See Section 2.6), where the objective function becomes the output of $\mathcal{NN}$. The



Candidate brackets
Generation i=0
Evaluation G0
Weighted probability
b_1^{can}
b_C^{can}
Ind 1
Ind M
b_k^{can}
$f_i = \frac{1}{O}\sum_{o=1}^{O} \max_{b_i^{can} \in \{b_1,...,b_k\}} S(b_i^{can}, b_o^{out}))$
$p_j = \frac{f_j}{\sum_{i=1}^{M} f_i}$
Creation of Gi+1
Outcome brackets
b_1^{out}
b_O^{out}
1- Conserve Z ind. with highest pj
2- For M-Z new ind, according to pj, randomly select two ind.
a. Conserve brackets present in both ind.
b. Complete new ind by randomly selecting brackets among the remaining ones
c. For y % of the new ind, randomly modify one bracket of the new Ind with another candidate bracket
Repeat for L generations or until convergence

Figure 3.2 Genetic algorithm

model for the single-entry transforms to

$$\max_{X} \mathcal{NN}(f_{1,1}, ..., f_{G,1}) \tag{3.4}$$

$$\begin{aligned}
\text{s.t.} \quad & x_{t,r,k} \leq x_{i,r-1,k}, && \forall\, t \in T, r \in R \setminus \{1\}, k \in \{1\} && (3.5)\\
& \sum_{t \in T(g)} x_{t,R(g),k} = 1, && \forall\, g \in G, k \in \{1\} && (3.6)\\
& \sum_{r=1}^{R(g)} 2^{r-1} \sum_{t \in T(g)} x_{t,R(g),k} Q_{t,R(g)} = f_{g,k} && \forall\, g \in G, k \in \{1\} && (3.7)\\
& x_{t,r,k} \in \{0,1\}, && \forall\, t \in T, r \in R, k \in \{1\}. && (3.8)\\
& && && (3.9)
\end{aligned}$$

In order to extend this model to a two-entry model, we need to train a new NN also denoted as $\mathcal{NN}$. This NN takes as argument the expected score of every selection of each entry and a binary set of variable which indicate the same selection among each entry. $\mathcal{NN}$ is then adapted to the multi-entry model introduced in Section 2.6 which transforms to

$$\max_{X} \mathcal{NN}(f_{1,1}, \ldots, f_{G,1}, f_{1,2}, \ldots, f_{G,2}, v_1, \ldots, v_G) \tag{3.10}$$

$$\begin{aligned}
\text{s.t.} \quad & x_{t,r,k} \leq x_{i,r-1,k}, && \forall\, t \in T, r \in R \setminus \{1\}, k \in \{1,2\} && (3.11)\\
& \sum_{t \in T(g)} x_{t,R(g),k} = 1, && \forall\, g \in G, k \in \{1,2\} && (3.12)\\
& \sum_{r=1}^{R(g)} 2^{r-1} \sum_{t \in T(g)} x_{t,R(g),k} Q_{t,R(g)} = f_{g,k}, && \forall\, g \in G, k \in \{1,2\} && (3.13)\\
& \sum_{t \in T(g)} 2s_{t,R(g)} - x_{t,R(g),1} - x_{t,R(g),2} \leq 0, && \forall\, g \in G && (3.14)\\
& \sum_{t \in T(g)} x_{t,R(g),1} + x_{t,R(g),2} - 2s_{t,R(g)} \leq 1, && \forall\, g \in G && (3.15)\\
& \sum_{t \in T(g)} s_{t,R(g)} - v_g = 0, && \forall\, g \in G && (3.16)\\
& x_{t,r,k} \in \{0,1\}, && \forall\, t \in T, r \in R, k \in \{1,2\}. && (3.17)
\end{aligned}$$

We modify the NN to the two-entry model introduced in Section 3.0.3 and add the linear constraints (3.14), (3.15) and (3.16) to ensure that

$$v_g = \begin{cases} 1 \text{ if both entries have the same pick in game } g, \\ 0 \text{ otherwise.} \end{cases}$$

CHAPTER 4 EXPERIMENTAL EVALUATION

In this chapter, we report on a collection of experiments that were conducted to evaluate the efficacy of the proposed models.

4.1 Computational platform

All experiments are conducted on an Intel(R) Xeon(R) Gold 6142 CPU at 2.60GHz with a limit 1 core. Source code and synthetic instances are available upon request.

This work is done using `Python 3.7.7`. In solving IPs, we use `Gurobi 9.0.0` [38], and utilize `JANOS 0.0.9` [33] for optimizing over NN-embedded optimization models. We learn the NNs using `Pytorch 1.4.0` [39] and convert the pytorch model into a Scikit-learn object by using `Scikit-learn 0.22.1`[34] which is required by `JANOS`. We also train the logistic regression by using `Scikit-learn 0.22.1`.

4.2 Probability models

In order to test the algorithms developed in this paper, we train a logistic regression model, denoted as **P1**, to predict the outcomes of the games. This model has the following features: the seed of both teams in the tournament and the ratio between those values. As discuss in Section 1, the *March Madness* is divided in four "regions", and the teams of each region are ranked from 1 being the best team to 16 being the worst team in the region. The lowest seed is the first parameter, so the ratio of both seeds is always less than or equal to 1. For every game between two teams with the same seed, we assign a .5 probability of winning the game to each team. For each year that we test the optimization algorithms, we train a model based on data for the *March Madness* tournament for every year between 2002 and the year immediately preceding the tournament. Table 4.1 reports the performance of the seed-based model on every *March Madness* tournament played since 2002, for both the men's and women's tournaments.

As can be seen in Table 4.1, the accuracy of this simple model provides reasonable probability estimates for games in the *March Madness* tournament. Predicting the outcome of a *March Madness* game is challenging due to the high variance in the results from year to year. Although more advanced models can be designed, this seed-based model has proven to be a competitive model empirically throughout the years (e.g., [40, 41, 9]). However, the purpose of this paper is to introduce optimization models for team selection, and any probability

Table 4.1 **P1** model performance for year's 2002 to 2019

	Man P1			**Woman P1**		
year	Accuracy	ROC AUC	logloss	Accuracy	ROC AUC	logloss
2019	73.02%	0.83	0.50	82.54%	0.81	0.38
2018	70.97%	0.66	0.59	79.37%	0.80	0.44
2017	74.19%	0.62	0.52	80.95%	0.73	0.45
2016	69.35%	0.68	0.59	76.19%	0.82	0.48
2015	73.77%	0.58	0.52	80.95%	0.73	0.39
2014	61.90%	0.65	0.63	80.95%	0.75	0.44
2013	69.35%	0.65	0.61	74.60%	0.77	0.46
2012	70.97%	0.64	0.57	85.71%	0.56	0.41
2011	65.08%	0.63	0.61	74.60%	0.87	0.45
2010	67.74%	0.70	0.58	74.60%	0.66	0.47
2009	73.02%	0.77	0.49	77.78%	0.76	0.48
2008	75.00%	0.63	0.52	80.95%	0.81	0.38
2007	83.87%	0.86	0.43	76.19%	0.70	0.50
2006	63.49%	0.70	0.59	80.95%	0.69	0.42
2005	74.19%	0.74	0.54	79.37%	0.83	0.43
2004	71.43%	0.70	0.52	71.43%	0.75	0.54
2003	69.84%	0.83	0.51	80.95%	0.75	0.40
2002	67.74%	0.58	0.57	82.54%	0.74	0.44
Average	71.00%	0.69	0.55	78.92%	0.75	0.44

model can be directly applied.

We also observe that the seed-based model is a better predictor for the women's tournament than the men's tournament. This can be in part attributed to the increased likelihood of upsets in the men's tournament (see Table 4.6 and Table 4.7).

4.3 Calculation the Expected Value of the Maximum

The first challenge is to identify a mechanism by which we can efficiently calculate, exactly or approximately, the expected value of a two bracket solution (i.e., the expected value of the highest scoring of the two). The tree-based algorithm is an exact method, but unfortunately takes extremely long to solve.

We compare in Figure 4.1 the exact approach with the SAA approach. To produce the figure, we first simulate an evaluation set of 280 two-bracket solutions, and compared the value obtained by the tree-based algorithm after 5 minutes of computation with the value obtained by the SAA, which is approximating the expected value of the two-bracket solution. For the SAA, for each of the 280 solutions, we calculate the average of the maximum score of the two brackets when evaluated on 50 randomly generated outcome brackets, repeated 100 times. We record the average and the standard deviation of the mean score of the maximum over these 100 repetitions.

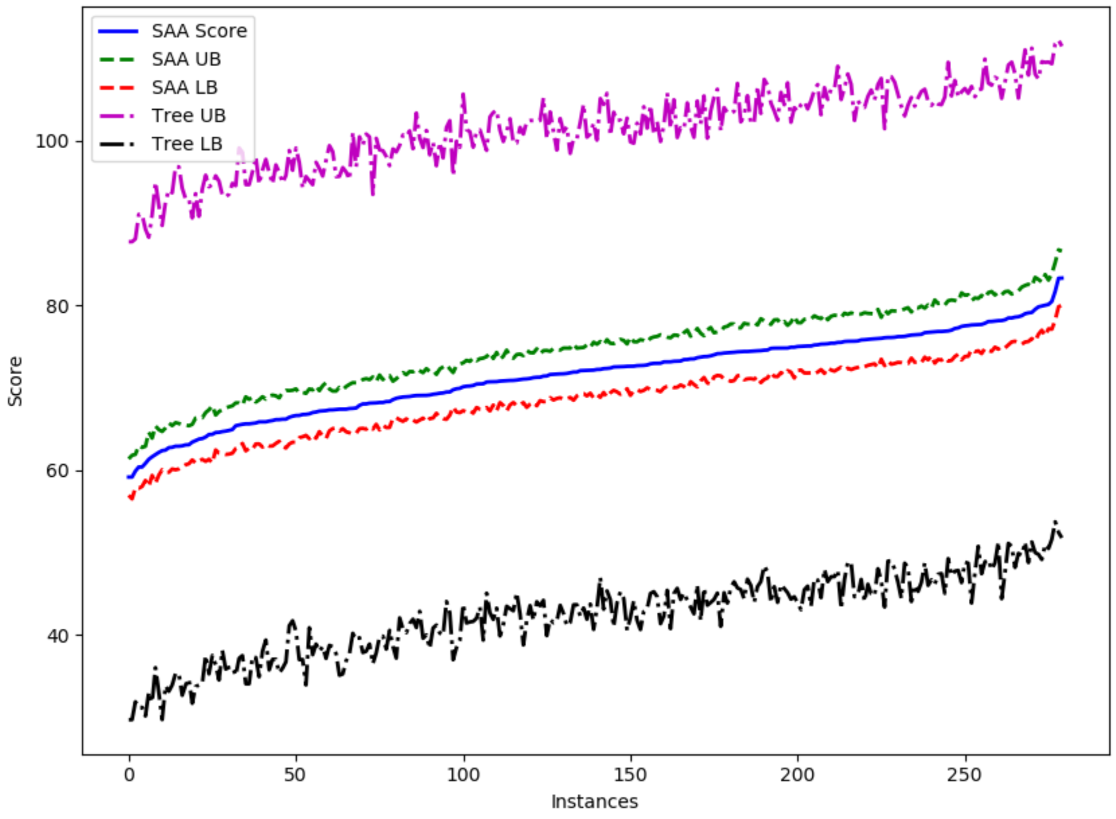


Figure 4.1 SAA vs Tree-based algorithm

Figure 4.1 depicts the results, sorted by the mean of the estimation. We depict in purple and black the upper and lower bounds obtained by the tree-based model within 5 minutes. In blue we depict the mean of the average scores over the 100 trials. In green and red we depict one standard deviation above and below the mean.

We see that SAA has a relatively tighter bound at approximating the expected value than the tree-based model we design. We therefore use simulation to approximate the expected value of a solution rather than an early stop of the tree-based model.

4.4 Optimization results

We now present results obtained by employing the variable optimization strategies discussed in this thesis. We first report on one-bracket solutions, and then on two-bracket solutions.

4.4.1 Single-entry evaluation

In order to test the potential of the solution approach, we first apply the NN-embedded model to the single-entry problem. This allows us to evaluate its effectiveness as a modeling paradigm, and to ensure the NNs generalize well when optimized over.

An NN is trained to estimate the expected value of a single bracket, denoted by $E(S(B_i))$. For the results we present, we trained a NN with two hidden layers with 32 units on each layer, with rectified-linear activation functions (this was used throughout so that it was ameable to the solver `JANOS`). We used MSE as the loss function, and we trained the NN using SGD with a learning rate of 0.01 and a momentum of 0.9. To regularize the NN, we implemented early stopping with a maximum of 15 epochs.

We use *Mean Absolute Percentage Error* (MAPE),

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\frac{|E(S(B_i)) - E\widehat{(S(B_i))}|}{E(S(B_i))},$$

to evaluate the quality of the prediction, where N is the size of the testing set, i indexes the set of solutions in our test set, B_i is the single bracket in the ith solution and $S(B_i)$ is a random variable representing the score of the solution's bracket i. The NNs are trained and validated respectively on 160,000 and 20,000 randomly generated single brackets. For both the training set and the test set, we generate random brackets through the simulation procedure defined in Section 3.0.2. $E(S(B_i))$ is the actual expectation of $S(B_i)$, which in the case of a single bracket can be computed exactly. $E\widehat{(S(B_i))}$ is the estimated expectation of $S(B_i)$ computed via the NN. To further support the accuracy of the NN predictions, we also report

$$\mathbf{A}(h) = \frac{1}{N}\sum_{i=1}^{N} I[|E(S(B_i) - E\widehat{(S(B_i))}| \leq h].$$

which represents the proportion of instances in the test set for which the prediction of the expected value of the solution given by the NN is within h units of the actual expectation.

The NN has high-quality prediction, with less than 1% MAPE and predicting an expectation within 1% of the expectation in over 99% of cases.

Table 4.2 Accuracy of NN for estimating the expectation of one entry

	Man P1			**Woman P1**		
Year	Training Time	MAPE	**A**(1)	Training Time	MAPE	**A**(1)
2019	942.01	0.0016	99.96%	857.96	0.0021	99.87%
2018	935.12	0.0013	99.88%	871.58	0.0021	99.86%
2017	958.27	0.0022	99.95%	876.88	0.0028	99.91%
2016	966.81	0.0010	99.91%	879.58	0.0017	99.97%
2015	997.11	0.0011	99.94%	905.69	0.0021	99.90%
2014	954.95	0.0014	99.93%	872.31	0.0017	99.97%
2013	914.45	0.0016	99.95%	894.71	0.002	99.96%
2012	908.66	0.0018	99.92%	868.83	0.0027	99.95%
2011	916.08	0.0013	99.91%	866.59	0.0021	99.87%
2010	938.19	0.0016	99.85%	855.34	0.0026	99.81%
2009	915.98	0.0014	99.92%	902.17	0.0018	99.94%
2008	920.95	0.0017	99.92%	892.72	0.0029	99.90%
2007	919.00	0.0021	99.90%	845.83	0.0023	99.83%
2006	913.12	0.0016	99.89%	876.51	0.0016	99.87%
2005	940.68	0.0012	99.95%	874.77	0.0020	99.92%
2004	969.52	0.0016	99.98%	940.99	0.0021	99.90%
2003	972.85	0.0015	99.98%	816.06	0.0022	99.89%
2002	969.25	0.0019	99.91%	953.95	0.0024	99.88%
Average	941.83	0.0016	99.92%	880.69	0.0022	99.90%

Equipped with a high-quality predictive model, we then use `JANOS` to find the best single-entry solution according to the prediction of the trained NN. In Table 4.3, we report the value of the optimal solution found by `JANOS` denoted as $\texttt{JANOS}^{1}$ (which is a bracket with an objective function that is an estimate of the expected value of the bracket), the exact value of the solution identified by `JANOS`, denoted as $\texttt{JANOS}^{\texttt{exact-1}}$, and the optimal solution found by the $\texttt{IP}^{\texttt{MM}}$ (which is the optimal value of the problem).

The purpose of this experiment is to evaluate how effective `JANOS` is at generalizing beyond the training set and finding a high-quality solution for the single-entry problem. Although the single-entry problem can be solved to provable optimality effectively by other approaches, evaluating the efficacy of `JANOS` in this setting suggests that it can also be applied to more general settings, such as multi-entry.

`JANOS` works very well for the single-entry setting. The predicted expectation of the best solution identified is nearly identical to the actual expectation of that solution. Furthermore, the quality of the actual expectation of the identified solution is on average (across the men's and women's tournaments from 2002-2019) only 0.31 points below the optimal value. These results validate that our predictive model integrated with the prescriptive model can find a

Table 4.3 JANOS solution quality for single-entry problem

	Men P1			Women P1		
Year	JANOS^1	JANOS$^{\text{exact-1}}$	IP$^{\text{MM}}$	JANOS^1	JANOS$^{\text{exact-1}}$	IP$^{\text{MM}}$
2019	79.37	79.49	79.52	98.43	98.44	98.59
2018	79.36	78.73	79.48	97.83	97.80	98.30
2017	78.95	79.00	79.28	97.84	97.99	98.13
2016	79.32	79.36	79.39	99.14	99.11	99.43
2015	79.31	79.29	79.38	97.32	97.49	97.49
2014	80.14	80.23	80.28	96.27	94.56	96.50
2013	80.74	80.70	80.86	95.44	95.54	95.84
2012	80.57	80.75	80.86	94.64	94.66	94.92
2011	83.26	83.33	83.41	94.58	94.58	94.95
2010	85.73	85.81	85.94	95.88	95.52	96.09
2009	82.85	82.90	83.00	96.86	96.97	97.09
2008	80.58	80.68	80.81	95.34	95.03	95.62
2007	79.33	79.21	79.61	97.81	97.14	97.90
2006	80.30	80.27	80.41	97.70	97.55	97.86
2005	79.77	79.85	79.89	96.16	95.79	96.23
2004	76.98	77.19	77.19	101.16	100.47	101.15
2003	76.20	76.24	76.27	102.12	101.85	102.24
2002	79.64	79.73	79.92	100.59	100.15	100.64
Average	80.13	80.15	80.31	97.51	97.26	97.72

near-optimal solution for the single-entry problem.

In Figure 4.2 we report the time to solve the various models previously proposed and introduced in the paper for the single-entry problem. We only report the results for the men's tournament as it is nearly identical for the women's tournament. In particular, for each year, we report the time to solve (a) the IP$^{\text{MM}}$ introduced in this work, (b) the DP presented by Kaplan et al. [37], and (c) the NN-embedded model in JANOS.

We see from the plot that IP$^{\text{MM}}$ is an order of magnitude faster than the previous DP model. Additionally, the complexity of optimizing over a NN with JANOS is clearly more challenging, and requires more computational effort. Despite being a challenging optimization model, the accuracy of the heuristic is quite good. We now extend it to the multi-entry problem.

4.4.2 Multi-entry evaluation

We now report on the application of our decision-making framework on the two-entry problem. Training an accurate NN for the multi-entry problem is much harder than for the single entry. Designing an analytical model for the two-entry problem appears to be intractable.

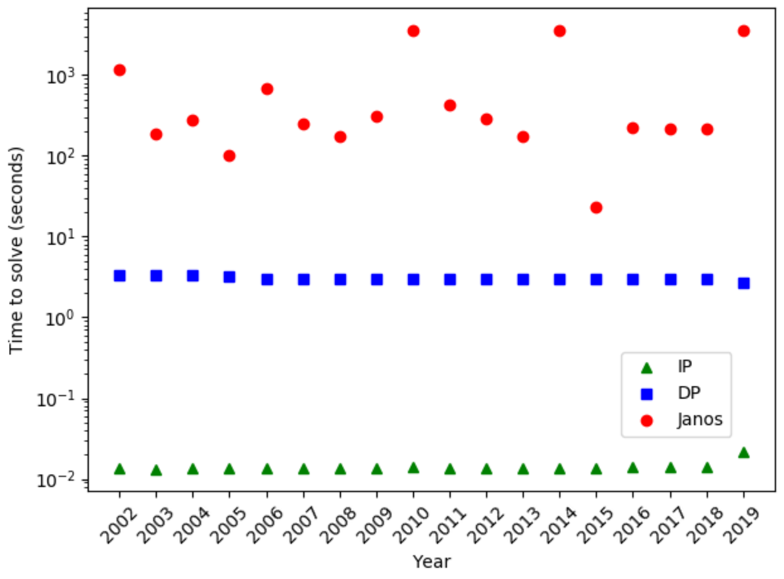


Figure 4.2 Time (seconds) to solve each instances for the men tourneys

We increase the size of the network to three layers with 32 units on each of the layers for the men's model, and four layers with 32 units on each layer for the women's model. For both models, we kept the same training parameters as the single-entry model, but use 300 epochs. Similarly to the single-entry problem, we report the computing time and the MAPE (which in this case can only be approximated due to lack of closed-form calculation), evaluated as:

$$MAPE = \frac{1}{N}\sum_{i=1}^{N} \frac{\left|\overline{E\left(\max_{k=1.....K} S(B_{i,k})\right)} - E\left(\widehat{\max_{k=1.....K} S(B_{i,k})}\right)\right|}{\overline{E\left(\max_{k=1.....K} S(B_{i,k})\right)}}$$

where N is the size of testing set, i indexes the set of instances in our test set, K is the number of entries, $B_{i,k}$ is the kth solution bracket of the ith instance, and $\max_{k=1.....K} S(B_{i,k})$ is the maximum score of the solution brackets in the ith instance. The NNs are trained and validated respectively on 160,000 and 20,000 randomly generated K-entry solution brackets for the men's tournament and 240,000 and 30,000 randomly generated K-entry solution brackets for the women's tournament. Since we can't compute exactly $E(\max_{k=1.....K} S(B_{i,k}))$, we estimate the value by evaluating the solution on 100,000 randomly generated outcome brackets and determine the maximum scoring bracket for each instance. Similarly to the single-entry case, we also report

$$\mathbf{A}(h) = \frac{1}{N}\sum_{i=1}^{N} I[|\overline{E(\max_{k=1.....K} S(B_{i,k}))} - E(\widehat{\max_{k=1.....K} S}(B_{i,k}))| \leq h]$$

which is the proportion of instances in the test set for which the prediction from the NN of the expected value of the maximum score of the K solution brackets is within h points of the simulated estimate. We report results in Table 4.4 with $K = 2$, $h = 1$, and $N = 20,000$ for the men's tournament and $N = 30,000$ for the women's tournament.

Table 4.4 Accuracy of NN for estimating $E(\max_{k=1.....K} S(B_k))$

	Man P1			**Woman P1**		
Year	Training Time	MAPE	$\mathbf{A}(1)$	Training Time	MAPE	$\mathbf{A}(1)$
2019	3003.89	0.0044	97.80%	5752.09	0.0028	99.63%
2018	3072.12	0.0044	97.74%	5770.77	0.0027	99.61%
2017	3164.24	0.0045	97.63%	5751.70	0.0029	99.58%
2016	3079.16	0.0047	96.96%	5902.01	0.0028	99.51%
2015	2870.31	0.0046	96.99%	5798.63	0.0028	99.59%
2014	2898.67	0.0047	96.95%	5891.28	0.0028	99.78%
2013	3452.96	0.0049	95.78%	5707.41	0.0030	99.50%
2012	3364.61	0.0044	97.30%	5732.65	0.0033	99.17%
2011	3243.01	0.0047	96.35%	5785.56	0.0030	99.26%
2010	3449.45	0.0048	94.62%	5859.99	0.0035	99.09%
2009	3245.77	0.0044	97.81%	5819.77	0.0033	99.22%
2008	3322.67	0.0049	95.94%	5775.52	0.0034	99.13%
2007	3028.29	0.0045	97.87%	5074.43	0.0056	97.50%
2006	2968.59	0.0046	97.06%	6396.26	0.0034	98.89%
2005	3116.60	0.0048	96.39%	6230.00	0.0036	98.76%
2004	2972.52	0.0041	98.93%	6650.47	0.0032	99.07%
2003	2992.12	0.0038	99.52%	6400.89	0.0033	98.42%
2002	2727.97	0.0043	98.01%	6674.33	0.0036	97.53%
Average	3109.61	0.0045	97.20%	5946.86	0.0033	99.07%

The complexity in learning the expected score of two brackets requires more training time than for the single entry. We observe that the accuracy of the men's model is smaller than the one for the women. However, the MAPE of both models is similar to that of the single-entry NN model. The difference in the accuracy is partly explained by the difference in the size of the NN and the training time.

We limit `JANOS` to 3600 seconds, and this time-limit is hit for all years. For `GA`, we initialize with 2,000 random single-brackets, use a population with 200 individuals for a maximum of

200 generations with again a time limit of 3600 seconds. We reach this time limit in every year.

In Table 4.5, we report two values for JANOS: the estimated expectation in the solution obtained denoted by $\texttt{JANOS}^2$ and the simulated expected score of the best solution JANOS finds, denoted by $\texttt{JANOS}^{\texttt{sim-2}}$. For GA, we simply report the estimated expectation for the best solution found, denoted by $\texttt{GA}^2$.

Table 4.5 Comparison of quality of solutions between JANOS and GA

	Man P1			**Woman P1**		
Year	$\texttt{JANOS}^2$	$\texttt{JANOS}^{\texttt{sim-2}}$	$\texttt{GA}^2$	$\texttt{JANOS}^2$	$\texttt{JANOS}^{\texttt{sim-2}}$	$\texttt{GA}^2$
2019	91.03	90.38	87.03	106.08	105.35	103.60
2018	90.53	90.44	85.83	105.35	103.00	103.14
2017	90.88	90.21	87.46	105.04	103.61	102.57
2016	89.55	89.59	86.79	107.05	105.02	104.18
2015	90.25	90.54	86.51	103.44	101.77	101.51
2014	91.75	91.27	87.45	97.66	92.66	101.55
2013	90.62	92.04	88.62	101.43	100.90	101.78
2012	92.70	92.14	88.36	101.65	100.24	98.70
2011	95.34	95.35	90.52	102.38	102.23	100.11
2010	96.34	96.07	95.15	98.47	98.13	100.61
2009	95.84	95.14	90.35	96.75	95.70	100.43
2008	92.30	91.99	86.69	102.79	102.05	100.88
2007	91.28	90.94	87.54	91.63	91.04	102.30
2006	92.30	91.89	88.58	106.28	105.34	102.20
2005	91.75	90.92	87.50	103.75	102.85	100.38
2004	84.44	83.32	80.09	103.03	102.69	106.05
2003	82.62	81.39	77.83	104.86	103.07	111.94
2002	89.97	90.27	85.89	112.87	111.74	108.78
Average	91.08	90.77	87.12	103.47	101.50	102.85

For the men's bracket, we see that the values estimated by the NN-embedded are very close to what results from simulating outcome brackets and evaluating the accuracy, namely a difference of 0.3 and 2.0 points on average for the men's and women's tournaments, respectively. We also note that in all cases, the quality of the solution we obtain through JANOS is better than what the GA model can determine based on evaluation over a simulated set of solutions. The solution determined by JANOS is estimated at 3.65 more points better than the solution identified by GA.

On the other hand, we observe that JANOS and GA find approximately the same expected value, on average, for the women's tournament. This diminishing quality of JANOS can perhaps be attributed to a lower variance in previous outcomes in terms of seeds. With fewer upsets, the women's bracket is harder to generalize, and perhaps more difficult to improve upon a more basic heuristic. This characteristic makes it more difficult to train the NN. Additionally, since high-quality solutions will contain fewer upsets than in the men's bracket, initializing `GA` with high-quality solutions is easier and therefore makes it more competitive with `JANOS`.

Note that the expectation of the two-entry solution improves substantially on the expectation of the single-entry solution. Compared with the solutions found by `JANOS`, the uplift in expectation is 10.28 points for the men's bracket and 4.24 for the women's bracket.

Finally, Table 4.6 and Table 4.7 reports results obtained by each model on each *March Madness* between 2002 and 2019. Results indicate that `JANOS` and `GA` are both able to find good solutions when tested on actual outcomes.

Table 4.6 Actual score obtained by every entry in the men's tournaments

Year	$\texttt{JANOS}^1$	$\texttt{IP}^{\texttt{MM}}$	$\texttt{JANOS}^2$	$\texttt{GA}^2$	# upset
2019	79	80	94	125	20
2018	113	115	110	118	20
2017	81	80	92	158	14
2016	88	89	86	86	20
2015	87	85	126	109	12
2014	67	66	69	68	22
2013	79	80	80	65	20
2012	86	86	95	92	17
2011	56	55	56	59	20
2010	86	86	87	85	20
2009	106	106	130	100	16
2008	112	114	141	119	13
2007	107	111	143	149	12
2006	60	61	65	63	21
2005	69	70	101	110	19
2004	64	64	63	66	16
2003	73	74	77	73	21
2002	125	125	109	82	16
mean	85.50	85.94	95.78	95.94	17.72
Geo.	83.33	83.69	92.37	91.85	17.41

For the men's tournament, all the scores of our best solutions remained far from the winning scores in the *ESPN Tournament Challenge*. It can be explained by the decision to use a seed-based model that strongly selects favorites to win each game of *March Madness*, which results in an inability to predict upsets. With an average of over 17 upsets per year in the men's

Table 4.7 Actual score obtained by every entry in the women's tournaments

Year	JANOS[1]	IP^{MM}	JANOS[2]	GA[2]	# upset
2019	88	89	91	108	11
2018	103	105	119	131	15
2017	69	70	73	89	12
2016	107	110	110	107	19
2015	132	132	148	132	10
2014	112	113	109	136	14
2013	111	108	110	117	14
2012	140	138	138	132	7
2011	76	78	79	77	18
2010	134	132	111	120	12
2009	83	82	126	120	16
2008	45	45	53	61	15
2007	61	62	94	111	14
2006	106	109	109	104	10
2005	98	99	98	91	17
2004	89	90	90	83	17
2003	163	166	162	144	10
2002	112	113	165	110	13
mean	101.61	102.28	110.27	109.61	13.56
Geo.	97.22	97.93	106.32	107.19	13.16

tournament, it would have perhaps been fruitful to use a probability model that recognizes more upsets. We can also observe that both JANOS and GA models obtained a similar average score that is 10 points above the single-entry model. On the 36 instances tested, JANOS outperforms GA 20 times while GA beats JANOS 15 times. As the probability model suggested, there are a smaller number of upsets in the women's tournament (on average, 4.16 fewer upsets per year) leading to an easier prediction problem for the women's tournament. Overall, we observe a very good score for the single-entry model in the women's tournament in 2003, reaching 166 points. It is also impressive that the GA is able to reach 158 points in 2017 in the men's tournament knowing that there were 14 upsets that year. We expect that increasing the number of entries and choosing a probability model that relies less on the seed will find more upsets that will lead to improved solutions. Similarly to Hunter et al. [18], selecting solutions with high variance and high expected value should be the ultimate goal of this optimization problem.

CHAPTER 5 CONCLUSION, FUTURE WORK AND RECOMMENDATIONS

5.1 Summary of work

This work introduced new approaches to the problem of evaluating and selecting multiple entries in sports betting contests. In particular, we study the *March Madness* tournament. This thesis presents two high-quality heuristics that optimize the expected value of order statistics, and that can potentially be expanded to more general settings. It is also among the first implementations of `JANOS` in the academic literature, and it explores NN-embedded models for which a lot of questions remain open.

More precisely, this work aims at identifying optimal brackets for a portfolio of entries in a sports betting tournament. We first explore different approaches for evaluating the objective function, which is particularly challenging. We present three approaches, including an exact tree-based algorithm and two approximation methods based on SAA and NN.

To optimize with this complex objective function, we then present two algorithms. The first is a `GA` and the second is a NN-embedded approach, which uses the recently introduced solver `JANOS`.

The results indicate that our optimization framework can be effective at identifying high-quality solutions. We report an improvement of 5% and 2% in expectation for the men's tournament and women's tournament, when moving from a single bracket to two brackets. When tested on real data, the men's model improvement remains the same, but the two-entries model are on average 4% better than the single-entry model. By increasing the number of brackets, we are optimistic that this framework would continue to improve upon the quality of the solutions, which we leave as future work.

5.2 Limitations

The biggest limitation for the `GA` is that it finds the best k bracket solutions consisting only of the candidates from a pre-populated set C. In order to be computationally feasible, C cannot be too large, since SAA over the sample outcome brackets requires significant computation effort.

The NN-embedded optimization model overcomes this challenge, and allows for an exhaustive search over all feasible solutions. Still, there are some limitations. For example, `JANOS` takes a long time to solve the optimization models, and significant effort is required for training the NNs. As the number of entries increase, we believe that `JANOS` will have a better capacity to find near-optimal due to its capacity to perform an exhaustive search.

5.3 Future research

For this work, it would be informative to learn how many entries are needed to achieve the winning score in the *ESPN Tournament Challenge*, which contains millions of entries submitted by participants throughout the world. Another possible extension is to apply more advanced predictive models for game outcomes, and see how the results differ.

Additionally, it would be interesting to see how our learning-based approach extends to different objective functions, like maximizing payoff or maximizing a combination of expectation and variance.

Finally, an intriguing observation on NN-embedded optimization that was observed in our experimental results was that as the solving time of `JANOS` increased, the evaluation of the solution found by the solver tended to decrease in quality (the objective function of the optimization model improved, but when it was evaluated by the simulation model the objective value decreased). This can be attributed to the challenge of building NNs that generalize well. A possible approach to counter this issue would be to use early stopping, similarly to how it is used in deep learning. We leave this for future research.

REFERENCES

[1] *Murphy v. National Collegiate Athletic Association, No. 16-476, 584 U.S. ____ (2018).*

[2] A. G. Association. (2020) State of the states 2020. [Online]. Available: https://www.americangaming.org/resources/state-of-the-states-2020/

[3] C. Gough. (2019) Sports betting and gambling market/industry - statistics facts. [Online]. Available: https://www.statista.com/topics/1740/sports-betting/

[4] Americain Gaming Association. (2019) Americans will wager $8.5 billion on march madness. [Online]. Available: https://www.americangaming.org/new/americans-will-wager-8-5-billion-on-march-madness/

[5] Wayne Staats. (2019) Ncaa bids 2019: Bracket for march madness. [Online]. Available: https://www.ncaa.com/news/basketball-men/2019-03-21/ncaa-bids-2019-bracket-march-madness

[6] Rob Wile. (2014) Warren buffett will give you $1 billion if you fill out a perfect 'march madness' bracket. [Online]. Available: https://www.businessinsider.com/warren-buffett-billion-dollar-bracket-2014-1

[7] Tom Huddleston Jr. (2019) The winner of warren buffett's march madness office pool could get $1 million a year for life. [Online]. Available: https://www.cnbc.com/2019/03/21/inside-warren-buffetts-multimillion-dollar-march-madness-challenge.html

[8] Kaggle. (2020) Ml competition 2020-ncaam - apply machine learning to ncaa march madness. [Online]. Available: https://www.kaggle.com/c/google-cloud-ncaa-march-madness-2020-division-1-mens-tournament/overview

[9] S. B. Caudill, "Predicting discrete outcomes with the maximum score estimator: The case of the ncaa men's basketball tournament," *International Journal of Forecasting*, vol. 19, no. 2, pp. 313–317, 2003.

[10] A. Hoegh *et al.*, "Nearest-neighbor matchup effects: accounting for team matchups for predicting march madness," *Journal of Quantitative Analysis in Sports*, vol. 11, no. 1, pp. 29–37, 2015.

[11] H. Ji *et al.*, "Matrix completion based model v2. 0: Predicting the winning probabilities of march madness matches," *Proc. of MSVESCC 2016*, 2016.

[12] J. Gumm, A. Barrett, and G. Hu, "A machine learning strategy for predicting march madness winners," in *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD).* IEEE, 2015, pp. 1–6.

[13] Y. Hao, B. Kristal, and D. F. Hsu, "Predication of ncaa bracket using recurrent neural network and combinatorial fusion," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech).* IEEE, 2018, pp. 897–903.

[14] P. Kvam and J. S. Sokol, "A logistic regression/markov chain model for ncaa basketball," *Naval Research Logistics (NrL)*, vol. 53, no. 8, pp. 788–803, 2006.

[15] A. Metrick, "March madness? strategic behavior in ncaa basketball tournament betting pools," *Journal of Economic Behavior & Organization*, vol. 30, no. 2, pp. 159–172, 1996.

[16] M. Wright and J. Wiens, "Method to their march madness: Insight from mining a novel large-scale dataset of pool brackets," in *KDD Workshop on Large-Scale Sports Analytics*, 2016.

[17] D. Bergman and J. Imbrogno, "Surviving a national football league survivor pool," *Operations Research*, vol. 65, no. 5, pp. 1343–1354, 2017.

[18] D. S. Hunter, J. P. Vielma, and T. Zaman, "Picking winners in daily fantasy sports using integer programming," *arXiv preprint arXiv:1604.01455*, 2016.

[19] M. B. Haugh and R. Singal, "How to play strategically in fantasy sports (and win)," in *Conference proceedings*, 2018.

[20] B. Clair and D. Letscher, "Optimal strategies for sports betting pools," *Operations Research*, vol. 55, no. 6, pp. 1163–1177, 2007.

[21] L. A. Wolsey, *Integer programming.* John Wiley & Sons, 1998, vol. 52.

[22] E. Osherovich. (2010) A convex function illustration. [Online]. Available: https://commons.wikimedia.org/wiki/File:ConvexFunction.svg

[23] Sdo. (2006) Shows a linear programming polytope together with a possible path (red) taken by the simplex method to solve the corresponding lp. [Online]. Available: https://commons.wikimedia.org/wiki/File:Simplex-method-3-dimensions.png

[24] Sdo. (2006) Polytopes of all feasible integer points and of the lp relaxation to the integer linear program max $\{\ y| - x + y \leq 1; 3x + 2y \leq 12; 2x + 3y \leq 12; x, y \in Z_+\ \}$. the green line is the cutting plane $x+2y \leq 6$ which cuts off the optimal lp solution (blue). [Online]. Available: https://commons.wikimedia.org/wiki/File:Cutting_plane_algorithm.png

[25] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition.* Ieee, 2009, pp. 248–255.

[26] D. Silver *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[27] O. Vinyals *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[28] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media, 2009.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[30] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *arXiv preprint arXiv:1811.06128*, 2018.

[31] E. Khalil *et al.*, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 6348–6358.

[32] M. Gasse *et al.*, "Exact combinatorial optimization with graph convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 580–15 592.

[33] D. Bergman *et al.*, "Janos: An integrated predictive and prescriptive modeling framework," *arXiv preprint arXiv:1911.09461*, 2019.

[34] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network flows," 1988.

[36] Bergman, David *et al.*, "Optimization for maximizing the expected value of order statistics," 2020.

[37] E. H. Kaplan and S. J. Garstka, "March madness and the office pool," *Management Science*, vol. 47, no. 3, pp. 369–382, 2001.

[38] L. Gurobi Optimization, “Gurobi optimizer reference manual,” 2020. [Online]. Available: http://www.gurobi.com

[39] A. Paszke *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach *et al.*, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[40] B. L. Boulier and H. O. Stekler, “Are sports seedings good predictors?: an evaluation,” *International Journal of Forecasting*, vol. 15, no. 1, pp. 83–91, 1999.

[41] J. G. S. S. Fonseca, “March madness prediction using machine learning techniques,” Ph.D. dissertation, 2018.

APPENDIX A MILP MODEL WITH SIMULATION MODEL

Another approach explored is to adapt the mutli-entry MILP model (2.39)-(2.47) to leverage the simulation-based model. We will denote this model as $\texttt{IP}^{\texttt{sim-2}}$. The reformulation goes as

$$\max_{x,h,s,z} \sum_{b\in\mathcal{B}'} \frac{h[b]}{|\mathcal{B}'|} \tag{A.1}$$

$$\begin{aligned}
\text{s.t.}\quad & x_{t,r,k} \leq x_{t,r-1,k} && \forall\, t \in T, r \in R \setminus \{1\}, k \in \{1,2\} && \text{(A.2)}\\
& \sum_{t\in T(g)} x_{t,R(g),k} = 1 && \forall\, g \in G, k \in \{1,2\} && \text{(A.3)}\\
& \sum_{g\in G} 2^{R(g)-1} x_{b[g],R(g),k} = s_b^k && \forall\, b \in \mathcal{B}', k \in \{1,2\} && \text{(A.4)}\\
& s_b^1 - Mz[b] \leq s_b^2 && \forall\, b \in \mathcal{B}' && \text{(A.5)}\\
& s_b^1 \geq s_b^2 - M(1 - z_b) && \forall\, b \in \mathcal{B}' && \text{(A.6)}\\
& h_b \leq s_b^1 + M(1 - z_b) && \forall\, b \in \mathcal{B}' && \text{(A.7)}\\
& h_b \leq s_b^2 + Mz_b && \forall\, b \in \mathcal{B}' && \text{(A.8)}\\
& x_{t,r,k}, h_b \in \{0,1\} && \forall t \in T, r \in R, k \in \{1,2\}, b \in \mathcal{B}. && \text{(A.9)}
\end{aligned}$$

where the only difference from (2.39)-(2.47) is the objective function. In this model, $\mathcal{B}'$ is a subset of randomly generated brackets of size M. In other words, this model summarizes to finding exactly the two brackets that maximize the expected value of the maximum scoring entry when evaluated on M randomly generated brackets.

Throughout this thesis, the SAA was always done using 100,000 randomly generated brackets. $M = 100,000$ appears to be to large to solve this model since the objective function and the number of constraints depends on M. Therefore, we tested this model using $M = 10,000$. In order to compare the solution given by this approach, we report for the men's tournament in Table B.1 the expected score of the maximum scoring entry of the two optimal brackets on another 100,000 randomly generated brackets to properly compare it with all the other approaches, and we report the actual points it obtained in Table B.2.

APPENDIX B SEQUENTIAL INTEGER PROGRAMMING MODEL

The last approach we tested was a Sequential Integer Programming (SIP) model denoted as `SIP`$^{\mathtt{k}}$ for the k-entry model. Rather than trying to maximize the expected score of the maximum scoring entry, this SIP model is focusing on finding good single-entry brackets without completely considering the dependency between each entry as we did in all other approaches. Therefore, the sequential approach sequentially add constraints such that we have at most σ picks identical in the last 4 rounds with all previously selected entries. This model goes as

$$\max_{x} \sum_{g \in G} \sum_{t \in T(g)} 2^{R(g)-1} Q_{t,R(g)} x_{i,R(g)} \tag{B.1}$$

$$\begin{aligned}
\text{s.t.} \quad & x_{t,r} \leq x_{t,r-1}, && \forall t \in T, r \in R && \text{(B.2)}\\
& \sum_{t \in T(g)} x_{t,R(g)} = 1, && \forall g \in G && \text{(B.3)}\\
& z_{t,r,k} - x_{t,r} \leq 0, && \forall t \in T, r \in \{2,3,4,5\}, k \in K' && \text{(B.4)}\\
& z_{t,r,k} - x^k_{t,r} \leq 0 && \forall t \in T, r \in \{2,3,4,5\}, k \in K' && \text{(B.5)}\\
& x_{t,r} + x^k_{t,r} - z_{t,r,k} \leq 1 && \forall t \in T, r \in \{2,3,4,5\}, k \in K' && \text{(B.6)}\\
& \sum_{t \in T} \sum_{r \in \{2,3,4,5\}} z_{t,r,k} \leq \sigma && \forall k \in K' && \text{(B.7)}\\
& x_{t,r,k} \in \{0,1\}, z_{t,r,k} \in \{0,1\} && \forall t \in T, r \in R, k \in K' && \text{(B.8)}
\end{aligned}$$

where

$$x_{t,r} = \begin{cases} 1, \text{ if team } t \text{ wins in round } r, \\ 0, \text{ otherwise,} \end{cases}$$

K' represents the number of solutions found by the SIP so far and x^k represents the k^{th} solution found by the SIP. Moreover,

$$z_{t,r,k} = \begin{cases} 1, \text{ if team } t \text{ is selected to wins in round } r \text{ in } x \text{ and } x^k, \\ 0, \text{ otherwise.} \end{cases}$$

This SIP is computed for K iteration where K is the number of entries. The SIP is exactly the same as the IP with the added constraints (B.4), (B.5), (B.6) and (B.7) which aims at increasing the variance of points obtained by each of the entries. Of course, this heuristic do not find the expected score of the multi-entry solution, but by regrouping all the selected brackets and using the SAA with 100,000 randomly generated brackets, we report the expected value of this multi-entry solution. The main advantage of this solution is that it is extremely fast to generate. We report the expected score and the actual score of the SIP for $K \in \{2, 3, 5, 10, 20\}$ in Table B.1 and Table B.2, respectively. We only report result for the two last models on the men's tournament.

Table B.1 Expected score of $\texttt{IP}^{\texttt{sim-2}}$ and $\texttt{SIP}^{\texttt{k}}$ for the men's tournament

Year	$\texttt{IP}^{\texttt{sim-2}}$	$\texttt{SIP}^2$	$\texttt{SIP}^3$	$\texttt{SIP}^5$	$\texttt{SIP}^{10}$	$\texttt{SIP}^{20}$
2019	89.47	88.15	91.97	96.16	101.35	111.29
2018	89.68	88.08	91.68	96.00	100.80	110.42
2017	89.73	87.81	91.28	95.96	102.31	111.65
2016	89.83	87.92	91.47	96.06	102.38	111.83
2015	89.06	87.70	91.53	95.77	102.38	113.37
2014	90.53	88.99	92.50	97.30	103.64	113.23
2013	91.23	89.53	92.95	97.77	104.63	116.21
2012	91.56	89.38	93.10	97.66	104.54	115.12
2011	95.45	93.63	96.74	103.51	108.89	120.90
2010	98.78	98.80	96.33	106.06	118.56	128.10
2009	95.11	93.82	96.39	102.93	108.80	122.24
2008	91.46	90.69	93.49	99.99	105.11	117.91
2007	90.27	89.42	91.98	97.99	103.59	116.54
2006	90.94	90.31	93.02	99.43	104.82	117.64
2005	90.02	88.21	91.90	96.51	103.52	113.80
2004	83.46	80.26	79.71	82.07	87.89	97.10
2003	81.37	79.46	80.00	81.10	84.65	92.66
2002	89.26	88.47	91.52	96.32	100.72	111.49
mean	90.40	88.92	91.53	96.59	102.70	113.42

We first observe that the expected score of the maximum scoring entry of $\texttt{SIP}^2$ scores on average almost 2 points lower than JANOS-2 while $\texttt{IP}^{\texttt{sim-2}}$ scores .67 point lower than JANOS. However, it is surprising to observe that $\texttt{SIP}^2$ performs slightly better than JANOS on actual data. The most interesting observation is the fact that the exact approach outperforms all other two entries models by almost 5 points on actual data showing the importance of modeling the expected value of the maximum scoring entry. It is also motivating to observe that the expected number of points continues to increase as we increase the number of entries.

Table B.2 Actual score of $\texttt{IP}^{\texttt{sim-2}}$ and $\texttt{SIP}^{\texttt{k}}$ for the men's tournament

Year	$\texttt{IP}^{\texttt{sim-2}}$	$\texttt{SIP}^2$	$\texttt{SIP}^3$	$\texttt{SIP}^5$	$\texttt{SIP}^{10}$	$\texttt{SIP}^{20}$
2019	132	96	96	132	132	132
2018	106	115	115	115	115	131
2017	115	112	112	124	140	140
2016	89	89	89	89	89	101
2015	134	145	149	149	149	149
2014	66	66	66	66	70	74
2013	80	80	108	108	116	120
2012	86	86	114	114	118	150
2011	55	55	55	55	55	63
2010	86	86	86	110	122	122
2009	98	106	130	146	146	146
2008	154	114	154	154	158	158
2007	121	143	143	143	143	167
2006	85	69	61	69	89	89
2005	134	102	102	114	138	142
2004	64	64	72	72	72	90
2003	126	74	74	74	74	80
2002	79	125	125	125	125	125
mean	100.56	95.94	102.83	108.83	113.94	121.06