Examining Algorithms to Win the game Rummikub

MATH 818.01 Midterm Survey

Jiyoon Jeong

Abstract

This survey presents an overview of the possible algorithms that may be used to make an program that aims to win the board game, Rummikub.

1 Introduction

Rummikub is a number-based board game. The players do not know the tiles that other players have and strategies that they choose to play; that is, every player lacks information in the game theory during the game. Since there are 106 tiles and each player has only 14 tiles each initially, the player does not know which player has what tiles and what tiles are left. Besides, it is not necessary to give away every possible set on the board, so it is comparatively harder to predict other players' strategies or status than complete information game. Assume a situation that a player A has only finished one's initial meld but does not give away sets anymore and has comparatively many tiles like 23 tiles. That player may have no sets to put down but perhaps trying to complete as many sets as possible in advance. Due to the complex property of the game, devising a program that wins the game Rummikub is likely to require some special algorithm or approach. Before getting into that, this paper will cover case studiesthat solve or try to solve difficult but interesting games by using artificial intelligence (AI). Furthermore, the prior research of winning the game Rummikub in a way other than using AI will be covered as well.

The paper is organized as follows:
In section 2 we introduce basic concepts prior to explaining the types of game in terms of complete information. Next, in the section 3, we describe case study of AlphaGo developed by Google Deepmind. In Section 4, we describe what algorithms and learning were used to devise a Poker AI, Pluribus, to look over an example of incomplete information game AI. In Section 5, we introduce integer linear programming (ILP) to plan the Rummikub implementation using ILP in Section 7.

2 Complete and Incomplete Information Game

There are diverse classifications of games. To name a few, zero-sum and non-zero-sum game, perfect and imperfect information game, and n-person game. To come up with algorithms to win the game

Rummikub, it is meaningful to look over prior researches of similar kinds of games.

2.1 Nash Equilibrium

A strategy is called a Nash equilibrium if no player can do better by unilaterally changing their strategy, assuming that each player knows others' strategy. In the example of the Prisoner's Dilemma, there is a dominant strategy: both confessing. Confessing is not always the best strategy, but it is the best strategy if 'both' choose to confess. The Prisoner's Dilemma is described in the next subsection.

2.2 Utility Function

Definition.
Utility function represents the degree of satisfaction of an input, which consists of various alternatives. Hence it reveals the preferences of the alternatives. The value of an output is to be maximized.

From now on, we introduce the Prisoner's Dilemma in mathematical language.

The set of players is $\{P_1, P_2\}$, P_1, P_2 distinct.

The players are going to choose whether one will confess or deny, so there are two actions for each player, put it $A_1 = \{C, D\}, A_2 = \{C, D\}$ respectively. C represents confession and D represents denial.

If both players choose to confess, then they each of them goes to prison for 3 years. If both choose to deny, then each of them goes to prison for 1 year each. If one confesses and the other denies, then the former goes free and the latter go to prison for 6 years. This is described on the table.

The set of outcomes is $\{(C, C), (D, D), (C, D), (D, C)\}$.
In one player's point of view, s/he does not know what the other player will choose but knows that the other player is offered the same deal as written below. Prisoner's Dilemma is an incomplete information game because the players do not know how the other player will act. Its definition is introduced in the next subsection.

P1 / P2	Confess	Deny
Confess	(-3,-3)	(0,-6)
Deny	(-6,0)	(-1,-1)

The utility of the player P_i is $u_i(a_i, a_{-i})$ where a_i denotes the action of a player Pi, and a_{-i} denotes the action of the other player. Hence the result is as follows: $u_i(C, C) = -3, u_i(D, D) = -1, u_i(C, D) = 0, u_i(D, C) = -6 \ \forall i = 1, 2$.

2.3 Information Game

Definition.
Information is literally the information that is or is not given to each player on the game.

Definition.
Information is common knowledge if it is known to all the players, if each player knows that all the players know it, if each player knows that all the players know that all the players know it, and so forth ad infinitum. This is defined recursively.

Definition.
Complete information game is a game where each player is fully aware of the rules of the game and the utility functions of every player. That is, each player has the common knowledge.

If we look over the meaning of the concept 'complete' itself, we can say as the following. Complete is a nature not moving first, or her initial move is observed by every player In a game of incomplete information, the meaning of complete is In contrast, an incomplete information refers to situations where some of the elements of the game are not common knowledge. The game Rummikub is classified into an incomplete information game which is an opposite concept of a complete information game.

The concepts explained so far can also be demonstrated in mathematical language and this is dealt in many books of Game Theory, as such, reference Perfect is also a popular concept when classifying the game because it is the strongest informational requirement. To put it another way, every player knows the meaning and strategies of other players' play. Besides, incomplete games are also imperfect games. The game Go, Chess and Shogi which are played smartly by AlphaZero are all perfect information games. Starcrafts2 and Poker are imperfect information games.

3 Case Study: AlphaGo

As seen in Section 2, the game of Go is a complete information game. AlphaGo is consists of Deep Neural Network (DNN) and Monte Carlo Tree Search (MCTS). In this survey paper, we only go through MCTS.

List of the methods is as follows: Supervised Learning (SL) of policy network, Reinforcement Learning (RL) of policy network, RL of value network, Searching with policy/value network (which is MCTS).

DNN is trained by humans and by itself to make AlphaGo a smarter player and MCTS efficiently searches the best move to choose it. Policy network and value network are improved by the training process of DNN and rollouts–playouts or single game from the beginning to the end–are made to update winning scores of nodes in search tree.

The policy network chooses the best next move. Move changes the current state by choosing an action, and turn it into the next state. Both SL and RL are used to improve policy network–that is, to make policy network smarter hence it can choose the best move. SL learns strategies given by human experts while RL is learned by self-plays.

Value network calculates the probability of final win when the move is suggested by policy network.

Rollouts are done randomly as the third step of Monte Carlo simulations.

3.1 Monte Carlo Tree Search (MCTS)

1. Monte Carlo Method

This method is done in heuristic way, that is, it is empirical method using probability. The phenomenon of neutron collision is the background of this method. The neutrons collide with one another, splits apart, thereby expelling neutrons. Here, splitting is occurred under the conditions of certain probabilities so these probabilities should be clearly known. The whole process is iterated and dispersed, hence leads to inducing explosion.

The mainstream of this algorithm is the idea that even a deterministic mathematical problem can be solved in a heuristic method using random sampling. The procedure is as follows: generating random number and calculating probability of certain goal information by using random number. As the number of iteration goes to infinity–in reality, the number increases–, the output value gets closer to the goal information–for example, winning scores of certain path.

Simple examples are computing π, ratio of the circumference of a circle to its diameter and the Buffon Needle problem. The code of the first example is attached in this page. Think of a square with a side length of 1 and the circle inscribed in the square. Each area of a figure is 1 and $\frac{\pi}{4}$ respectively. Then we calculate the probability of the points that goes inside the circle. If there are sufficiently many points, then we can estimate that (the probability of the points in the circle) $\times 4 \approx \pi$. The code is written in python language.

```
import random

x = 0
y = 0
circle = 0
count = 0

while True:
    for i in range(0, 1000000):
        x = random.random()
        y = random.random()

        if (x * x) + (y * y) <= 1:
            circle += 1

        count += 1

    print(count, (circle/count) * 4, sep=": ")
```

The result is as follows:

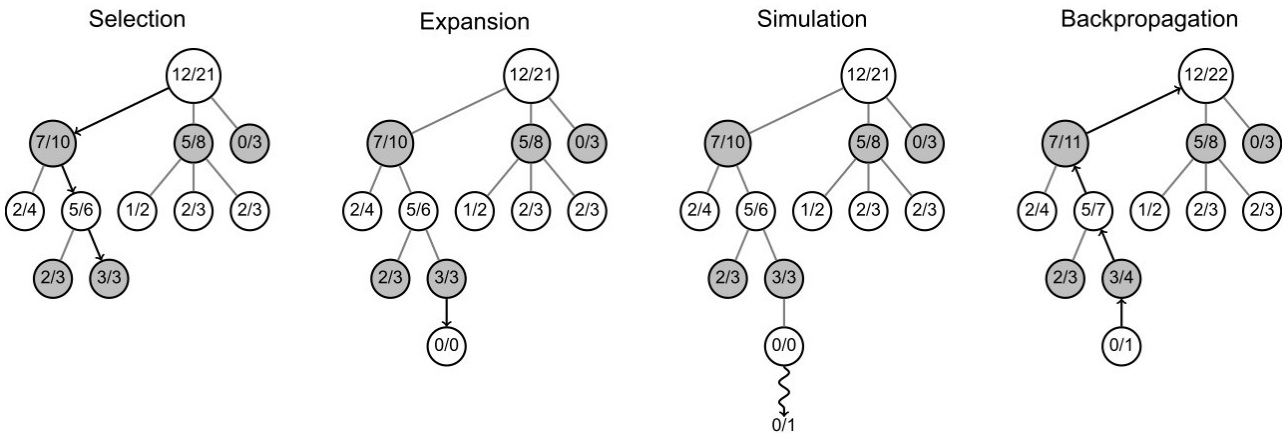


Figure 1: The diagram describing the process of MCTS

2. Monte Carlo Tree Search

The goal of this algorithm is to predict winning rates of paths of nodes without going through every single path. These days, MCTS is treated as a basic and prioritized searching algorithm after it was used in AlphaGo.

The process is as follows:

1) selection: In the selection phase, the algorithm starts at the root node and chooses a child node that has the maximum win rate at each node. It only extends to one further depth. The evaluation is done by Upper Confidence Bound applied to trees (UCT) formula:

$$\begin{aligned} UCT_i &= \frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}, where \\ w_i &= \text{number of wins after the i-th move,} \\ n_i &= \text{number of rollouts after the i-th move,} \\ c_i &= \text{exploration parameter and } c_1 = \sqrt{2}, \\ t &= \text{total number of simulations for the parent node,} \end{aligned} \tag{1}$$

The first term means the win rate and represents the exploitation and the second term exploration; we can see exploit-explore tradeoff in this formula. The former value is relatively higher value for moves with high average win rates. On the other hand, the latter value is higher for moves that undergone few simulations. The meaning of the second term... The UCT is theoretically proven to converge to MiniMax search, but practically, basic version of MCTS converges to MC Perfect game, which is not as thorough as MiniMax search.

MiniMax searching goes through every single node. If the implementation time takes too long, then traversal is gone through so far as the nodes in the certain depth. If this happens, then the result of the algorithm becomes far from the intended result.

2) expansion: In the expansion phase, one child node is expanded from the selected node.

3) simulation: In the simulation phase, simulation is the random rollout starting from the child node expanded in the former state which is generated by RL.

4) update(backpropagation): In the update phase, the algorithm goes back along the path to update the visit score and winning score of each node (or game state), thereby changing the UCT function. In this way, the node selection in the next round will be closer to win rates of winning tactics if they exist.

3. Advantages and disadvantages of MCTS

1) Advantages:

2) Disadvantages: Aside from the issues of a huge amount of memory and a speed. If the machine encounters unusual situation, then it chooses absurd decision. Since the search cannot cover every single case but covers only paths with the high win rate, some paths are gone through insufficiently. This phenomenon was shown in Game four of AlphaGo versus Lee Sedol (or the Google DeepMind Challenge Match). It is interpreted that Lee's unexpected play at white 78 would have made AlphaGo's poor respond on move 79, leading to Lee's win.

code for tic-tac-toe (only MCTS)

Alphazero also used MCTS to average the approximation errors made by non-linear function approximation, not alpha-beta pruning.

3.2 Alpha-beta Pruning

Alpha-beta pruning is a kind of search algorithm to cut down unnecessarily leaves as far as they do not affect the result. It is a kind of MiniMax searching.Alpha-beta search computes an explicit MiniMax, which propagates the biggest approximation errors to the root of the subtree. It requires a function to evaluate current state in every step, hence needs a great deal of computation of levels in the game tree to find the optimal move.

4 Poker AI

Poker is one of the most popular games that have a property of incomplete information. Many researchers strives to devise an AI that solves the Poker. There were several AIs of such, and one of them, Pluribus, is superhuman AI for multiplayer poker. DeepStack is AI of 2-player.

DeepStack uses the following algorithms.

Pluribus was shown to be stronger than top human experts in six-player no-limit Texas hold'em poker. In most cases to solve a game problem, finding Nash equilibrium is the usual goal. However, in this case, it is theoretically hard to find Nash equilibrium, so the researchers focused on finding

solutions in empirical means.

Things to consider:
1. As the number of players grow, it is computationally almost impossible to solve the game to the end.
2. A player's optimal strategy for a particular situation changes as other players' strategy is for every situation.

Used algorithms:
Pluribus is learned by only self-plays, without any prior data or strategies given. There are 3 representative methods and two of them are deep-limited searches.
First is the blueprint strategy computed using Monte Carlo counterfactual regret minimization (MCCFR) and Linear CFR. CFR is an iterative self-play algorithm that starts with random play to beat earlier versions of itself.
Second is real-time search. Real-time search is necessarily used in solving complete information game. MCCFR is used when the scale of computation is relatively small, but an optimized vector-based form of Linear CFR is used otherwise.

Abstraction is used to make the calculations scalable before applying the above strategies. Action abstraction reduces the number of actions. Here, making bet is an action, so only some selected bet sizes are used. If it becomes off-tree search, the algorithm finds the most similar target–that is, size. Information abstraction reduces the number of situations. Consequently, it drastically reduces the complexity of the game. Pluribus runs on a relatively small capacity of CPUs and memory. AlphaGo used 1920 CPUs and 280 GPUs for real-time search in 2016, but Pluribus used two Intel Haswell E5-2695 and uses less than 128 GB of memory.

5 Integer Linear Programming

Linear Programming (LP), or linear optimization, is an optimization problem in which the objective function is linear in the unknowns and the constraints consist of linear equalities and linear inequalities. LP is not a computer-based implementation, but only a mathematical method. For example, the simplex method is founded on the fact that the optimal value of a linear programis finite, is always attained at a basic feasible solution.

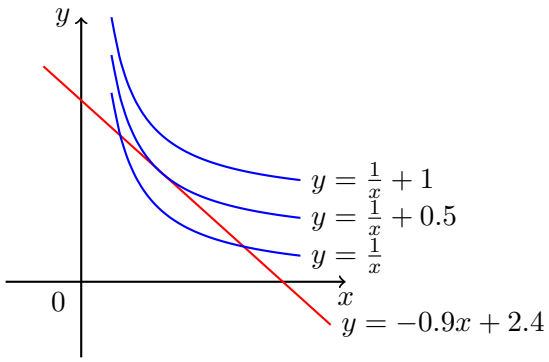
Optimization includes maximizing or minimizing a linear functional over a set of constraint polynomials. Linear equations can be reduced to a vector and matrix notation.

$\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{x}, \mathbf{c} \in \mathbb{R}^{\mathbf{n}}$, $\mathbf{b} \in \mathbb{R}^{\mathbf{m}}$

$$\begin{aligned} &\text{minimize } \mathbf{c^T x} \\ &\text{subject to } \mathbf{Ax} = \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}. \end{aligned} \tag{2}$$

Integer linear programming is a linear programming whose every entry of solution vector is integer.

Maximize the value of $y - \frac{1}{x}$ on the region surrounded by $x \geq 0, y \geq 0, y \leq -0.9x + 2.4$



Put $k = y - \frac{1}{x}$.

$y = \frac{1}{x} + k$ has the form of graph as illustrated as blue graphs. The graph should satisfy the following conditions: it should be on the region–some point on the graph should also be on the region–and it should make k maximum. Combining these two conditions, we can conclude that the graph should be as high as possible starting from x-axis, still meeting the region. Since $y = \frac{1}{x} + k$ has derivative $y' = -\frac{1}{x^2}$ on $(0, \infty)$, we can conclude that there exists some (x, y) such that $y' = -0.9$. That is, there exists k such that $y = \frac{1}{x} + k$ tangent to $y = -0.9x + 2.4$. By calculations, two graphs meet tangentially at the point $(\frac{1}{3}\sqrt{10}, -\frac{3}{10}\sqrt{10} + 2.4)$. By putting this value into $y = \frac{1}{x} + k$, we get the maximum $k = -\frac{3}{5}\sqrt{10} + 2.4$

We will see specific implementation of ILP in Section 7.

6 Basic rules of the game Rummikub

There are total 106 tiles–2 sets of tiles numbered 1 to 13 in four colors: black, red, blue, orange; 2 Jokers. There are 4 racks that the players can put their tiles on, hence 2 to 4 players can join the game. meld A set is Each players has a turn and when it is one's turn, that player give out sets or A group is a set of either three or four tiles of the same number in different colors. A run is a set of three or more consecutive numbers, all in the same color.

Scoring system

The game is finished when there is a player who has no tile left. The winner is determined by the score in the highest order: the player who first finished the rack wins and gets the score after calculating the scores of the others. Each player's score is the negative sum of the values of the tiles left. The winner's score is the sum of each player's sum of values of the tiles, which is positive. The Joker tile is counted as the value of 50 if it is on the rack. If the player did not completed the initial meld, then the score is -100.

7 ILP on Rummikub

There are two goals of this programming:
(1) The major goal is to maximize the value of the tiles that can be placed onto the table.
(2) The minor goal is to minimize the movements of the existing sets on the table.
It seems like this programming is a greedy algorithm if we only look over the major goal. However, in order to consider real practice, changing as small sets on the table as possible is more efficient in terms of time, energy, and memory like a human.

First, give the model:

Indices (I, J are mathematical sets)

$i \in I$ indicates the type of the tile (defined by color and number), $I = \{1, 2, \cdots, 53\}$
$j \in J$ indicates the index of the set (run or group), $J = \{1, 2, \cdots, 1174\}$

There are 2 copies of $13 \times 4 = 52$ tiles with 2 Jokers whose roles are indistinguishable. Hence there are 53 types of tiles. That explains the index set I.

To explain J, the number of Jokers should be taken into account.

Number of Jokers	0	1	2	Total
groups of 3	$13 \times 4 = 52$	$13 \times \binom{4}{2} = 78$	$13 \times 4 = 52$	182
groups of 4	13	$13 \times \binom{4}{3} = 52$	$13 \times \binom{4}{2} = 78$	143
runs of 3	$11 \times 4 = 44$	$(12 + 11) \times 4 = 92$	same case as 'groups of 3'	136
runs of 4	$10 \times 4 = 40$	$(11 + 10 + 10) \times 4 = 124$	$(12 + 11 + 10) \times 4 = 132$	296
runs of 5	$9 \times 4 = 36$	$(10 + 9 + 9 + 9) \times 4 = 148$	$= 233$	417

Total number of sets is 1174

1) The linear programming only reflects goal (1)

Parameters

s_{ij} indicates whether tile i is in set j (yes= 1, no= 0),
t_j indicates the tile i is $0, 1$ or 2 times on the table,
r_j indicates the tile i is $0, 1$ or 2 times on your rack.

Variables

x_j indicates that the set j can be placed $0, 1$ or 2 times onto the table,
y_i indicates that the tile i can be placed $0, 1$ or 2 from rack onto the table.

Objective and constraints

$$\max \sum_{j=1}^{53} v_i y_i \tag{3}$$

subject to

$$\sum_{j=1}^{1174} s_{ji} x_j = t_i + y_i \ \forall i$$

$$y_i \leq r_i \ \forall i$$

$$x_j \in \{0, 1, 2\} \ \forall j$$

$$y_i \in \{0, 1, 2\} \ \forall i$$

The meaning of the constraint is..

This model contains at most 1174 variables and 53 real constraints. Furthermore, note that this model is always feasible: ....

2) The next linear programming given includes goals (1) and (2).

Parameters

w_j set j is 0, 1 or 2 times on the table

M constant (a number more than 35, default 40)

Variables

z_j the difference between the number of occurrence of the set j in the new solution and the old solution.

Objective and constraints

$$\max \sum_{j=1}^{53} v_i y_i + \frac{1}{M} \sum_{j=1}^{1174} z_i \tag{4}$$

subject to

$$\sum_{j=1}^{1174} s_{ji} x_j = t_i + y_i \ \forall i$$

$$y_i \leq r_i \ \forall i$$

$$z_i \leq x_i \ \forall i$$

$$z_i \leq w_i \ \forall i$$

$$x_j \in \{0, 1, 2\} \ \forall j$$

$$y_i \in \{0, 1, 2\} \ \forall i$$

$$z_i \in \{0, 1, 2\} \ \forall i$$

If we only consider the condition (1), $\max \sum_{i=1}^{53} v_i y_i$ in which v_i is the value of the tile i.

There are at most $35 = \lfloor \frac{53 \times 2}{3} \rfloor$, because there are 106 tiles and the shortest possible length of a set is 3.

The ILP model is implemented in AIMMS (Advanced Integrated Multi-Dimensional Modelling Software) on a Pentium III

8 Planning Implementation

There are 4 goals of this programming:
(1) The major goal is to maximize the number of value of the tiles that can be placed onto the table.
(2) Another minor goal is to prioritize two numbers $13, 12, 1, 2$ in this order.
(3) Use the Joker only if there is no other set.
(4) One minor goal is to minimize the movement of the existing sets on the table.

From now on, the reasons why such goals were suggested be explained intuitively with examples.

To explain (2), intuitively, the number $13, 1$ and $12, 2$ have comparatively less possibility to form a set and since their values are among the highest, these numbers should be put down as soon as possible. We cannot calculate every single case of certain set being on the table in advance. Hence we only briefly consider the probability of being part of a set as follows:
$13, 1$: $\frac{}{1174}$ $12, 2$: $\frac{}{1174}$

To explain (3), even if using the Joker tiles is the only way, a human player does not necessarily put down one's set. The player may wait until suitable sets appear without using the Joker tiles. The strategy depends on the player, but (1) contradicts with the strategy to make the best use of the Joker tile; that is, to use the Joker in the 'best' situation. For example, assume that the player A has the tile (mathematical) set $\{2(black), 4(black), 7(orange), 8(orange), Joker\}$. According to this program, using the Joker is the only way to put down the set. $\{7(orange), 8(orange), Joker\}$ or $\{Joker, 7(orange), 8(orange)\}$, $\{2(black), Joker, 4(black)\}$ are the only ways. However, a human player may be looking forward to obtain $3(black), 6(orange), 9(orange)$ in one's turn because the possibility to complete the runs is not very low compared to the situations like below.

However, assume this case: $\{1(black), 5(black), 7(orange), 8(orange), Joker\}$. How would a player act? In order to reduce the total value of the rack, the player would probably follow the goals suggested above. Or, others would choose to obtain new tiles to complete black tiles–1,2,3,4,5–or orange tiles–6,7,8 or 7,8,9. There is a big assumption to consider: the present state. What if there are already some tiles on the board? What if those tiles are not to be utilized; that is, unchangable in the present status?

Hence, it is very hard to make all situations into consideration.

In my personal point of view the minor goal suggested by the previous section is a lot more minor than the other goals suggested above. Since it has value less than value of any tile, its weight is very low. Hence we use the second term of the formula (2)

(4) One minor goal is to minimize the movement of the existing sets on the table.

Computer of a high capacity is (or are) needed in order to implement this ILP.