Dynamic Greedy Algorithm for Big Two Card Game's AI

Matthew - 13507012

Informatics Engineering
School of Electrical dan Informatics Engineering
Bandung Institute of Technology
Ganesa Street 10
E-mail: matt_jcs@yahoo.com

ABSTRACT

Big two or sometimes known as *Cap Sah* (used in Hokkien and Indonesia) is one of most popular card games in the world. Usually this card game is played by four people (although two people still enough to play the game). Recently there's big two game played in computer or phone application so the player can play with other player or with the AI (Artificial Intelligence) from the application.

Even there's some AI for big two games, but a lot of AI is still too easy to be defeated, and sometimes the AI still have bug on it, making the player easier to win the game. This paper will represent a dynamic greedy algorithm, that will make a very powerful (a.k.a. hard level) AI in big two games. The word 'dynamic' here means the AI can decide how the greedy algorithms will work which will be used next turn with noticing the game environment.

The keys to win the games will be based on the author great experience of playing big two games and based on some online literatures. Also on this paper, the author will explain the game rules, the trick to win the game, and how those points implemented in AI's algorithm.

Keywords: big two, card game, artificial intelligence, dynamic greedy algorithm, powerful AI.

1. INTRODUCTION

Big two or sometimes called "Chinese Poker" is a card game, played with two to four players, and 13 cards for each player (sometimes the entire deck being dealt out depends on the number of participant player). The objective of the game is to spend all cards in the hand. When one of the players runs out of cards, the game is over and the rest card on the other players' hand will be counted as the score.

Figure 1. 4 player card game, each 13 cards

The card rank is its name, that 2 is the highest card, followed by A K Q J 10 9 8 7 6 5 4 and the lowest value is 3; the suit rank *is* ♠ (Spades) > ♥ (Hearts) > ♣ (Clubs or Clovers) > ♦ (Diamonds). Therefor the highest card in the game is 2♠ and the lowest is 3♦.

The card can be played as singles or in groups of two, three, or five, in combinations which resemble poker hands. The leading card to a trick sets down the number of cards to be played; all the cards of a trick must contain the same number of cards. The combinations and their rankings are as follows:

- **Single cards**: Any card from the deck, ordered by rank with suit being the tie-breaker. (For instance, A♠ beats A♦, which beats K♠.)
- **Pairs**: Any two cards of matching rank, ordered as with singular cards by the card of the higher suit. (A pair consisting of the K♠ and K♦ beats a pair consisting of K♥ and K♣.)
- **Three of a kind**: Any three cards of matching rank, ordered by rank, two's rank high, as usual.

- **5-card hand**: There are five different valid 5-card hands, ranking, from low to high, as follows:
  - **Straight**: Any 5 cards in a sequence (but not the entire same suit). Rank is determined by the highest valued card with the suit used only as a tie-breaker. Therefore 2-3-4-5-6 > 10-J-Q-K-A. However, A-2-3-4-5 is lowest possible straight for strategic purposes.
  - **Flush** (also called **flower**): Any 5 cards of the same suit (but not in a sequence). Rank is determined by highest suit, and then by highest rank card.
  - **Full House**: a composite of a three-of-a-kind combination and a pair. Rank is determined by the value of the triple, regardless of the value of the pair.
  - **Four of a kind + one** card (nicknamed **Bomb**): Any set of 4 cards of the same rank, plus any 5th card. (A 4 of a kind cannot be played unless it is played as a 5-card hand) Rank is determined by the value of the 4 card set, regardless of the value of the 5th card.
  - **Straight Flush**: A composite of the straight and flush: five cards in sequence in the same suit. Ranked the same as straights, suit being a tie-breaker.

2. RULES AND TRICKS

Big two have so many variant rules (as other popular card games). At this paper, the author will explain the most common rules and also represent some tricks that can make the AI stronger.

2.1 RULES

a. At the beginning of each game, the player with the 3♦ starts by either playing it singly or as part of a combination, leading to the first trick. Play proceeds counter-clockwise, with normal climbing-game rules applying: **each player must play a higher card or combination** than the one before, with the same number of cards.

b. Players may also **pass**, thus declaring that he does not want to play (or does not hold the necessary cards to make a play possible). A pass does not hinder any further play in the game, each being independent, referred to as jumping-back.

c. When all but one of the players **have passed** in succession the trick is over, and the cards are gathered up and a new trick is started by the last player to play. When a player plays the 2♠ either as a single or as part of a pair of 2s, it is often customary for that player to start the next trick immediately by leading a new card or combination, since the 2♠ cannot be beaten whether as a single or as part of a pair of 2s, and the passes are mere formalities.

d. Scoring varies from place to place. The most common version is that after a game each player with cards remaining scores -1 point for each, unless they have 10 or more remaining, in which they score -2 for each. If they didn't get to play any cards at all, they score -3 for each. Then the winner of the hand scores +1 for every -1 his opponents got. (So, for example, if North won, and East, West, and South respectively still had 3, 11, and 8 cards left, East would score -3, West would score -22, South would score -8, and North would score +33.)

2.2 TRICKS

The first trick to win the game is to settle the best card combination. However, in the real game, the card must be resettled every turn, by considering the leading card set, count other players hands, and count of player card count.

Second is to remember cards that have been thrown to the board. This part is very important, but hard to do with human memory. By remembering the entire card that has come out, the player can predict the highest card now.
For example, if 2♠,2♦,A♥,A♣,A♦,K♥,K♦ has come out, and North have 2♣,K♣,Q♠,*Q*♦, this situation make North's Q pair the strongest pair (because no other pair stronger than that for now). Knowing this will be a great benefit for the player in deciding what card is safe / best to be thrown.

Third one is considering the score. Even the best player in the world can't win in every big two games. Therefor a good player must know how to reduce the minus point before he lost the game. This trick can be applied by considering the cards count in the hand, and the cards count in the other players' hand.

The last trick is the greedy. Without knowing the meaning of greedy, many great players use this 'greedy' to win the big two games. The greedy in this context talk about the worst card set to be thrown as soon as possible at **that turn**, and keep the best for the late. This greedy appear because it's a lot easier to throw good card such as K♦ A♣ 2♥ that to throw 3♠ 4♥ or 5♣, and it's easier to throw a full house of 3♠3♥3♣4♠4♣ than to throw a straight of 9♥10♣J♠Q♦K♣. And considering the condition, throwing a 5-card hand is the best choice if the other players only have 1 card, 3cards, and 4 cards (of course other players can't beat even just throw the lowest straight).

3. THE GREEDY

In a greedy algorithm, the optimal solution is built up one piece at a time. At each stage the best feasible candidate is chosen as the next piece of the solution. There is no back-tracking. The generic greedy algorithm must contain these elements:

1. A set C of candidates
2. A set S of selected items
3. A solution check: does the set S provide a solution to the problem (ignoring questions of optimality)?
4. A feasibility check: can the set S be extended to a solution to the problem?
5. A select function which evaluates the items in C (select the best candidate).
6. An objective function (the real final goal)

The schema is like this in the general:

```
// a greedy pseudo code written in C++ code
// a function with a set of C (candidate),
return a set of S(solution)
set Greedy (set C) {
       // declaration variable
       candidate x;
       set S; // a set of candidate that
                 represent the solution

       initialize(S);
       while (!Solution(S) && !EmptySet(C))
       {
              x = Selection(C) //choose a
                           candidate from C
              C = C - x  // pop the x
                         element from C
              if (Feasible (S+x)) {
                     S = S + x; // S is add
                         with the element x
              }
       }
       if (Solution(S))
       {
              return S;
       }
       else
       {
              cout << "There's no solution
                        found";
              return S; // return empty set
       }
}
```

3.1 DYNAMIC GREEDY

Unlike the ordinary greedy, the author will use dynamic greedy for implementing the algorithm on the big two AI. The dynamic greedy definition here is the elements of the greedy aren't static, but dynamic. It means that the candidate, the solution check, the feasibility check, and the select function always changing for each use, but of course the objective function always the same.

The dynamic greedy become powerful than the ordinary greedy, because the ordinary greedy always have the same parameter, but the dynamic one can change the parameter to reach the best goal (objective function). Even that, the dynamic greedy must be more complicated and having more complexity at its algorithm.

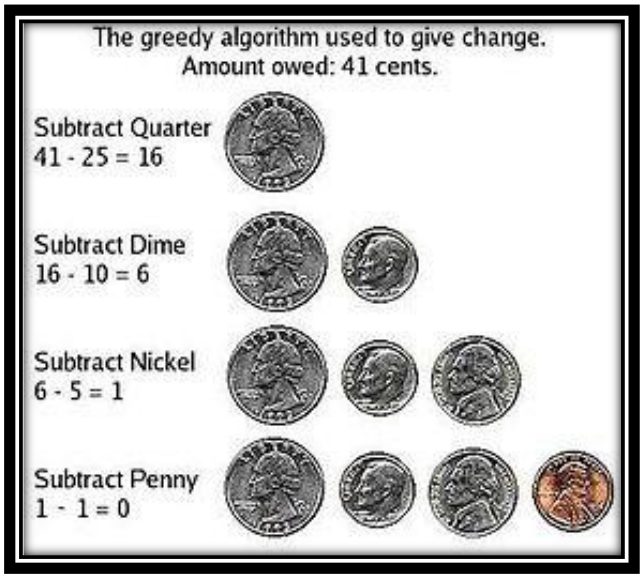

Figure 2. Greedy algorithm for change diagram

The most classic case for greedy algorithm is the minimum coins in change problem at the vending machine (The coins usually are 1c, 5c, 10c, 25c, 50c). The greedy-way to get the minimum number of coins is repeatedly adding the largest coin that doesn't go over the change value.

The author here thinks about some other different cases that sometimes occur in the real world. It's not impossible that vending machine ran out the 5c and 10c coin. (It means the machine needs to change the select function that excludes 5c and 10c from the selection). And how if the user put in more coin after picked the drink can (it means the machine must change something else:
First credits – drink price + more coin = change the feasibility check?).

There's no doubt that the real world is too much complex if we want make all of them into algorithm. But for the general and essential problem, it's still possible to make some algorithm. These points can make the dynamic greedy algorithm more effective and reliable to use.

4. IMPLEMENT THE DYNAMIC GREEDY INTO BIG TWO'S AI

The dynamic greedy must be implemented into algorithm, so we can get the AI. The first thing to do is make the game without any AI, so we have the structure of the cards and the rules.

4.1 The Card Type Structure

The card structure will be a multidimensional array, with size 4 x 13.

```
// array represent of card deck
int masterDeck [4][13];
int northDeck [4][13];
int eastDeck [4][13];
int westDeck [4][13];
int southDeck [4][13];
```

The array [4] represent the card suit; array [0] for ♦, array [1] for ♣, array [2] for ♥, array [3] for ♠. And the array [13] is for the number order; array [0] is for number 3 (notice that 3 is the lowest card rank), array [1] is for 4, and so on, array [11] for A, and array [12] is for 2.

At the beginning, the master deck initialize by 1, other deck will be initialized by 0. After that, it will call method to share a card. Then it will randomize assign the 1 to the other deck.

Figure 3 Example a card set for North deck

Table 1 Array represent the North deck

	♦	♣	♥	♠
3	**1**	**0**	**0**	**0**
4	**0**	**1**	**0**	**1**
5	**0**	**1**	**0**	**0**
6	**0**	**0**	**0**	**0**
7	**0**	**1**	**0**	**0**
8	**0**	**0**	**0**	**1**
9	**0**	**0**	**0**	**0**
10	**0**	**0**	**0**	**0**
J	**1**	**1**	**1**	**0**
Q	**0**	**0**	**0**	**0**
K	**0**	**0**	**0**	**0**
A	**0**	**1**	**1**	**0**
2	**0**	**1**	**0**	**1**

The deck will be that figure, the 1 represent that North have that card. If North threw a full house 4♣4♠J♦J♣J♠, North's array at that index will be assigned by 0, and the masterDeck array at that index will be assigned by 1.

Besides the card type structure, the trick set combination need to be mentioned to. The valid trick set in big two is single card, pair, three kind of card, and five-card set. The rule is the climbing-game rule, so the requisite is the value of card trick is higher than the card in the ground now.

Single card

The single card is easy to differ, the value just see the position of the deck array index, first priority is the [13] array (represent the number), then the [4] array (represent the suit).

Pair card and Three of a kind

Same as single card the value just order by the card deck index.

Five-card hand

First thing is giving trick value, the straight full is 5; bomb is 4; full house is 3; flush is 2;straight is 1. After the trick value, then the tricks self must have own value. Straight will depend on the highest single value card of the straight. Flush depend on the suit first, then the card rank. Full house depends on the "three of a kind" value. Bomb depends on the "four of a kind" value. The straight flush depends similar like flush set.

4.2 Greedy Priority

The significant part of a greedy algorithm is to order priority of the candidates. Now as mention before at chapter 2.2, the algorithm here will be based on the tricks.

First is to settle the card combination. To settle the combination, it must consider these points:

1. The leading card trick (single, pair, three-kind, five-card set, or the owner will lead a new trick).
2. The other players' card count. One of this point purposes is to prevent losing big score.
   **point 1 and 2 make the greedy needs dynamic*
3. The owner deck, what tricks are available, how high is the card value according the rest card that have come out.
4. The value of set combination. (having five pairs of course better than a five-card set with five random and low value single card, but five-card set with five random and high value single card is much better than the five pairs).

To make those points more reality and possible to transform into algorithm, each card level, each trick set, and each deck combination will be given a greedy value (let's just say as ***GV***). The greedy only will calculate the current GV of the deck, and to decide what the AI will do next (what card, or trick set will throw this turn) is just simply calculate the entire possible move's GV. Then, the greedy will choose the move that save the biggest GV point.

4.2 GV Point Example

The GV example here is a summary from the trick on chapter 2.2. The GV is separated by the card count. So the GV will be like this.
Each card has value for being single card, pair card, and three card of a kind. While the value is dynamic, according the highest value card now (because the card which have been thrown is remembered by the AI)

Table 2 GV for single and pair card

Card number	GV for single				GV for pair	
Card Suit	♦	♣	♥	♠	♦♣♥	♠
3	-9	-9	-9	-9	20	20
4	-7	-7	-7	-7	25	25
5	-5	-5	-5	-5	30	30
6	-2	-2	-2	-2	35	35
7	0	0	0	0	40	40
8	1	1	1	1	45	45
9	2	3	4	5	50	50
10	6	7	8	9	45	45
J	10	11	12	13	50	50
Q	14	15	16	17	55	60
K	18	19	20	21	65	70
A	22	23	24	25	75	80
2	30	35	40	50	30	25

The pair 2 gets few GV because the two is better played as single card. Other than that, the GV is relative too, because the GV can rise if the higher card has been thrown.

If the AI gets the chance to lead a card trick, then this is the GV for each trick:
The GV to play single card (losing 1 card) is 0. GV for play a pair is 50 (losing 2 cards). GV for play a three of a kind is 50 (losing 3 cards). And GV to play five of a kind is 150 (losing 5 cards).

Next state is the AI has to follow the other players trick. This state is a bit hard to decide, because sometimes it needs to throw the card, but sometimes it need to just pass because if throw the card will break the combination set card. So the parameter won't be just single parameter, but some. In this example, the author chose 3 significant parameters. It's the leading trick card now, the AI's card count, other cards count. The other cards count especially; can be more specialized again, because we need to know who is having fewer cards, and whose leading the trick set now. To make the real best AI will need very detail GV, and making a table for it is a good options. But to describe the example here, the author only makes some description.

✓ **Leading card: single card, GV normal for single: 0**
Own card: 10-13
One of other player card: less than 7
GV to throw single card increased by 30
*reason: it's too dangerous having more than 10 card (can cause double minus score if lose the game.

Own card: less than 5
GV to throw single card increased by 20
*reason: having less 5 cards is easy to win by single card only and a bit hard to force win with throwing higher card first.

One of other player card: 1
GV to throw pair card increased by 20
*reason: that player who has 1 card can immediately win anytime.

Next player card: 1
GV change to throw the largest card is the best
*reason: the next player turn can win if we throw a low rank card.

✓ **Leading card: pair, GV normal for pair: 50**
Own card: 10-13
One of other player card: less than 7
GV to throw pair card increased by 30
*reason: it's too dangerous having more than 10 card (can cause double minus score if lose the game.

Own card: less than 5
GV to throw single card increased by 50

*reason: having less 5 cards and can still play a pair is very good option.

One of other player card: 2
GV to throw pair card increased by 200
*reason: in case that player has a pair, than it can win by throwing the remaining 2 cards.

✓ **Leading card: five card set, GV normal : 150**
One of other player card: 5 - 7
GV to throw pair card increased by 100
*reason: if the other players have one more set five cards, it can be so dangerous if the AI don't stop them.

The GV constant here is very simple. As mention before, to make a better GV set we must define it on a table. And each changing card count of the AI and others players must have different GV calculation. Other than that, the relative highest card can be counted into GV calculation too.

5. SUMMARY

As the above example illustrates, we know all the basic concepts to making the dynamic greedy AI for Big Two game. But from this basic paper, it's more easy for make more complex algorithm and powerful AI, because it mentioned here how to expand the complexity.

The greedy here will always take the 'best' move for that time, considering the dynamic parameter, such as the AI's card count, the other player card count, and the leading card trick now. The greedy in this AI chooses greedy by the biggest GV point. Because higher card have bigger GV, than the AI will always throw smaller card first, but not breaking the card combination; because the card combination have its own GV and the GV for combination is bigger than single card. The card combinations here are pair, tree of a kind, and five-card hand.

For better development, we need to make some test case and recalculate the GV point. And if required we can have more GV aspects for the AI

REFERENCES

[1] *Dariusz Kowalski*, "Lectures on Greedy Algorithms and Dynamic Programming", <http://www.csc.liv.ac.uk/~darek/COMP523/lecture-greedy-dynamic.ppt>, University of Liverpool, 2009.

[2] Rinaldi Munir, "Strategi Algoritmik", 2006, page 26-34.

[3] Wayne Goddard, "Introduction to Algorithms"< http://www.cs.clemson.edu/~goddard/texts/cpsc840/part2.pdf>, Clemson University, 2004, page 2-3.

[4] Wikipedia, Big Two, <http://en.wikipedia.org/wiki/Big_Two>, accessed on December 30, 2009.