Title

Pestalozzi Lukas

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Projet Semestre, 4^{th} Master

Supervised by

Igor Kulev and Boy Faltings

Spring 2017

Abstract

Tichu is a multiplayer card game with hidden information and both cooperative and competitive elements. Those properties makes it an interesting game from the point of view of Artificial Intelligence. In [4], David Pfander shows that it is possible to create an agent that is able to compete with an average human player. However, his approach relies on carefully crafted heuristics based on specific "expert" knowledge about the tactics of the game. This project explores ... without expert knowledge / general approach easy applicable to other similar games ... concentrate on how to deal with hidden information and big game tree ...
Results, knowing other hands is advanageous ... it is possible to a certain degree to learn from data of previous played games (human player games) ... computational budged is a problem ... "semi expert" knowledge helps ...

Contents

List of Figures

List of Tables

Introduction

AI computer perfect information games
what about hidden information?
Tichu is such a game

1.1 "Research Question"

1.2 Project Outcomes / Results

1.3 Project Overview

Implementation of the framework ?

The game Tichu

Tichu is a 4-player gambling card game which is marketed by Fata-Morgana [1]. It falls into the category of ladder games, where players must play a "higher" ranked set of cards on their turn or pass. The four players build 2 teams which compete against each other for points. Tichu contains elements from some Chinese card-games such as "Dou Di Zhu" or "Zheng Fen" and is very popular in the german part of Switzerland.

Add picture of the game

2.1 Rules

More detailed rules can be found in [1, 2].
The game is played in successive rounds until one team reaches a predefined amount of points.

Cards
There are 4 suits (Sword, Pagoda, Star, Jade) of 13 cards each corresponding in rank to the standard Bridge cards (2, 3, ... ,10, J, Q, K, A). In addition there are 4 special cards without any suit: *Mah-Jong, Dog, Phoenix* and *Dragon*. Which brings the total number of 56 cards.

Preparation
The winner of the previous round shuffles and cuts the deck. When all cards are distributed, each player gives one card to each of the 3 other players. This phase is called card trading.

The Game
A round is started by the player who has the *Mah-Jong* card. He may lay down any one of the following combinations:

Single A single card

Pair Two cards of the same rank

Triple Three cards of the same rank

Square Four cards of the same rank

Fullhouse A triple and pair together

Straight At least five cards of consecutive ranks

Pair Step Two or more consecutive Pairs (ie, J, J, Q, Q, A, A)

The next player (on the right) has two options, either to play a combination of the same type but with a higher rank, or to pass (not play at all). That means, for example, a pair can only be

beaten by another pair of strictly higher rank and a straight only by another higher straight of the same length.

The play then proceeds to the next player. If all players pass consecutively the trick ends and the player who laid the last combination takes the trick and leads a new one. If he has no more hand-cards, he retires from the game and the next not-retired player may start the trick.

Bombs
Bombs are either a straight where all cards have the same suit ("straight flush") or four cards of the same rank ("square"). A bomb beats all other combination and a higher bomb beats a lower one, (any straight bomb is higher than a square bomb and a longer straight bomb beats a shorter one). Bombs can be played at any time during a trick but after the first play was made.

The Special Cards
Mah-Jong: Whoever has this cards makes the first lead. The *Mah-Jong* has a rank of 1 and can be used in a straight (eg. 1,2,3,4,5,6). When a player plays the *Mah-Jong*, he can wish any specific card rank (not including special cards). The next player who has a card with the wished rank and can play it *in accordance with the rules of the game* must play it. This condition remains in force until somebody plays such a card.

Dog: The dog has no rank and can only be played as single card and only as *lead-play* (first card of a new trick). It immediately gives the lead to the partner of the player who played the dog (it can't be beaten by a bomb). If the partner already finished (has no handcards left) then the lead passes to the player on the right.

Dragon: The dragon is the highest single card and can also only be played as a single card. If the Dragon wins the trick, the player must give the entire trick to any player of the opposing team.

Phoenix: The phoenix can be used as a joker in any combination (but it can't be used to create a bomb). It also can't replace any other special card. When played as a single card it has a value half a point above the last played card but it can't beat the dragon. If it is played first it has a value of 1.5.

Calling Tichu
Before playing his first card, each player has the right to announce a "Tichu". If he then wins the round (finishes first), his team receives 100 additional points - otherwise the team loses 100 points. A player can also announce a "grand Tichu" before getting the 9^{th} card from the dealer. This gives 200 additional points (in case of success) or a penalty of -200 otherwise.

Goal of the game
The round is played until three of the four players have played all of their cards. The remaining player gives the remaining handcards to the opposing team and all the tricks he won to the player who finished first. Then the cards of each team are counted as follows:

- Kings and 10's are worth 10 points each
- 5's are 5 points each
- The Dragon is worth 25 points
- The Phoenix costs 25 points (is worth -25)

However, if both players of the same team have a double victory (both finished before any of the opposing team), then that team gets 200 points and the card points are not counted.

When one team reaches a predefined number of points (usually 1000) they are the winners of the game.

2.2 Tactics

- finishing first is most of the time better than looking at the points of cards during the game. - prevent an enemy double win is paramount - points won by doublewins and tichus often dominate points won with tricks (in human plays). - getting rid of cards in ascending order is already a good strategy. -¿ important to have high cards left at the end. - supporting the teammate is important, especially when called tichu (dog may be handy).

ono: drache am verlierer gä, zersch fertig isch guet wilme ono pünkt vom stapl bechunnt

2.3 Tichu and Artificial Intelligence

Tichu is quite different from "classical" games such as chess or ...

First of all, it is a game with hidden information since the handcards of the other players are unknown (with the exception of the traded cards at the beginning). This introduces several problems discussed in the section *Hidden Information*.

The second big difference to the classical games is that the game is both, a cooperative and competitive at the same time. In a team the players must play cooperatively and help each other while competing with the other team for points.

After the dealing of the cards, Tichu is a deterministic game. That is, the effect of each action is known with certainty. However, it is not possible to determine precisely what the possible actions of any of the other players may be since their handcards are not known. This makes is necessary to model Tichu as a stochastic game from the viewpoint of a player.

The game is played over several, in essence independent, rounds which makes it an episodic game. Each round consists of several tricks being played, so each round has also an episodic nature. However, unlike the rounds, the tricks of the same round are not independent. The actions in a trick depend on actions played in previous ones.

Tichu is a discrete, turn based game. The fact that bombs can be played at any time can be modeled with a 'fast round': After each played combination all players have to decide whether they want to play a bomb or not. Then, the next player can play a 'normal' combination.

Finally, Tichu is a zero-sum game even though the points of a round typically don't sum up to zero. At the end exactly one team wins, the other looses. It is possible to center the points after each round around zero to make them sum up to zero without impacting the gameplay at all.

2.3.1 Difficulties / herausforderungen

Tichu is in several aspects a "difficult" game for AI. In this section some of the difficulties are discussed and ...

Hidden Information The first problem with games with hidden information is that a player does not have all information necessary to find the optimal action. This leads to uncertainty about "how good" each action actually is since it depends, at least partially, on the hidden information (or else the game is probably not fun to play). In most such games it is a big advantage to learn or at least approximate the hidden information, Tichu is no exception. A player knowing exactly what the other players handcards are can determine the best action with 100% confidence. However, due to the random dealing of the cards, knowing everything doesn't guarantee a win. It may happen that the enemy just has better cards. This implies that perfect information might not be as important in Tichu as in other games.

There are roughly two different approaches to deal with hidden information. Infer the information, for example from the way a player plays, or deal with it by finding out what actions are good in a lot of possible cases. A big part of this project is dedicated to explore both approaches.

Big branching factor Even without the hidden information ... The game-tree of a typical Tichu game has two different node-types in regard to the branching factor. The *first-play* actions (a player can play any combination-type) have a branching factor that is typically between 10 and 30, but can reach up to 450 in rare cases. Especially in the beginning of the game, when all players still have most cards, the branching factors are relatively big. The nodes corresponding to the remaining plays in the trick have typically a small branching factor (seldom more than 3) since the player has to play the same combination-type as played in the *first-play*.

Most games go through 8-15 tricks, each with 10-20 actions in each. Thus, a game contains roughly 100 actions in total. (The longest possible game lasts for 220 actions (the leading player always plays a single card and the other players always pass such that the loosing player has 1 card left at the end $\leq 4 * 14 * 3 + 4 * 13 = 220$.)

Therefore, a typical game-tree is one that branches out rather fast near the root (at the beginning of the game), is around 100 deep and contains 8-15 *first-plays*. Leading to a tree with roughly 10^{20} nodes. From those numbers it is clear that even an exhaustive search for one particular deal is practically unfeasible.

Unlike in chess, where the starting position is always the same, in Tichu (practically) each game starts with a different deal. This excludes the possibility of creating the Tichu equivalent of an opening-moves database. Therefore each player has to compute the best action online.

Multiagent, cooperative and competitive The simultaneous cooperative and competitive nature of the game gives raise to other interesting challenges. However they are only addressed to a certain degree in this project.

The other big difficulty, which was not explored but equally interesting and important (see future work/ not addressed problems)

Implementation

Because there is no (to my knowledge) open framework for tichu and AI, I had to implement my own. python, whole framework, compatible with OpenAIgym to be able to use RL libraries

3.1 architecture

gamemanager, gamestate, cardset, combination, player, agent interface, environment interface agent gets gamestate and returns action. gamestate contains handcards of all players, so the agent can cheat if so desired. note that agents are "stateless" but could easily be expanded with methos such as "gamestarts" or "newround" etc...

mention not repeating same actions drops branching factor

For example in a Hand with the three kings $K\heartsuit, K\diamondsuit, K\clubsuit$ it is possible to play three different pairs of kings. However, it does not matter at all which two kings are played since the suit has no relevance. In this case only one out of the three possible pairs has to be considered as a possible action.
The only exception to this observation is when the hand contains a *straight-flush* containing a king. Then it does matter which king(suit) is played in order to not 'destroy' the *straight-flush*.

3.2 difficulties

finding architecture, determine all possible DIFFERENT combinations in a set of cards, performance (especially mcts),

Agents

In this section the different agents are introduced and discussed separately, and in the next section they are compared against each other.

Simplifications

To limit the scope of the project and to be able to concentrate on the 'card play' aspect of Tichu I decided to simplify the agents as follows:

No Trading The agents all trade random cards. This has the same effect as omitting the trading phase.

No Tichu announcement As mentioned in 2.2, points won with Tichus often dominate points won by wining tricks. This diverts from the 'card play' aspect, Therefore all Agents are implemented such that they never announce any Tichu

Bombs not anytime For simplicity and ease of simulation, a player can only play when it's his turn. That means bombs can not be played out of turn. This has not a big impact on the game play overall, since it is rare to gain an advantage in playing a bomb out of turn.

No Wish For a similar reason as the 'no Tichu' simplification, the agents do not wish any card.

4.1 Background

This section contains a short overview of the main algorithms used by the different agents.

An important notion is the **gametree**, which is a tree containing as nodes all possible gamestates and as edges the (game)actions leading from a parent state to a child state. The root is the initial state of the game, and the leafs are states in which the game ended (one player or team has won).

4.1.1 Minimax Search

Given a gametree, the optimal strategy can be determined from the minimax value of each node. This value denotes how 'good' a gamestate is for the searching player and is computed as follows:

$$minimaxval(s) = \begin{cases} Reward(s), & \text{if s is terminal} \\ \max_{a \in Actions(s)} minimaxval(next(s,a)), & \text{if player(s) = searching player} \\ \min_{a \in Actions(s)} minimaxval(next(s,a)), & \text{otherwise} \end{cases}$$

where s is a gamestate, $Reward(s)$ denotes the reward for the searching player in s, $Actions(s)$ is the set of legal actions in s, $player(s)$ denotes the player able to play in s and $next(s, a)$ is the resulting state when playing action a in s.
This algorithm performs a complete depth-first search of the gametree, which is computationally unfeasible in most games, but it is a good theoretical baseline for other algorithms.

Despite this, it is possible to use minimax in larger games by stopping at non terminal states and evaluating them with an *utility* function. Minimax has been successfully used in games where it is possible to create a good *utility* function, such as chess.

Furthermore, it is possible to prune away large parts of the searchtree without loosing accuracy using **alpha-beta pruning**. This method makes use of the fact that some subtrees don't influence the final result since the players never play suboptimal actions. For more details see [5, chapter 5, p. 170+] or any textbook about AI.

4.1.2 Monte Carlo Tree Search (MCTS)

Similar to minimax search, MCTS also searches the gametree, but uses simulated games to evaluate nonterminal states and thus has no need for an utility function. This is especially useful for games where it is hard to determine whether a state is 'good' or 'bad' for a player, for example in games with hidden information, but also in perfect information games such as GO.

The algorithm iteratively builds a subtree of the gametree where the root corresponds to the current state of the game. Each iteration consists of the 4 phases shown in Fig. 4.1:
Tree Selection The algorithm traverses the (in previous iterations built up) gametree until it reaches a leaf-node. The decision which actions to follow during the traversal is determined by the *tree policy*. The tree policy attempts to balance considerations of exploration (look in areas that have not been well sampled yet) and exploitation (look in areas which appear to be promising). An often used tree policy is the UCT (Upper Confidence bound for Trees) formula proposed by [13] (see section 4.4.1).
Expansion Once arrived at a leaf state, one more action is chosen and a node for the resulting gamestate is added to the tree. Form this state a *simulation* (or *rollout*) is performed.
Simulation A simulation is the run from a gamestate to a terminal state of the game. During simulation the moves are made according to the *default policy*, which typically chooses an action uniformly at random.
Backpropagation The final state is evaluated and the reward for each player is calculated. The reward is then used to update the statistics of the nodes visited during tree selection, which in turn is used for the *tree policy* in future iterations.

The search can be terminated at any point which makes MCTS especially suitable for games with time or computational resource constraints.

4.1.3 Determinization and Perfect Information MCTS (PIMCTS)

Determinization is the process of taking a gamestate with hidden information and creating ('determining') a perfect information state that is consistent with the hidden information. In essence, determinization chooses one out of the many possible perfect information gamestates for a given hidden information gamestate.

To deal with hidden information, PIMCTS performs several instances of MCTS, each on a different determinization. The best action is then chosen based on the results of the individual MCTS. This

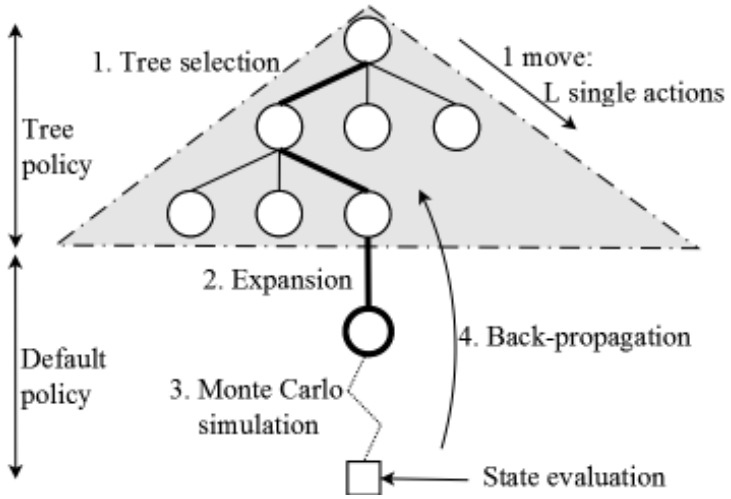


Figure 4.1: The basic steps of MCTS. Source [12]

approach was successfully used for the game Bridge in [14], but has two main drawbacks, described in [14] and further explored in [3] and [7]:
Strategy fusion: An agent can only make one decision in each hidden information state, but in different determinizations different decisions can be made based on the perfect information. This lets the agent 'believe' that he can distinguish between determinizations of the same state, which is not the case. Strategy fusion can cause the agent to choose a bad over a winning action.
Nonlocality: Some determinizations may be vanishingly unlikely (rendering their solutions irrelevant to the overall decision process) due to the other players' abilities to direct play away from the corresponding states.

4.1.4 Information Set MCTS (ISMCTS)

To tackle the problem of strategy fusion, *Cowling et al.* propose *Information Set MCTS* in [7]. Instead of building a different searchtree for each determinization, ISMCTS builds one search tree with the information from all determinizations using *Information Sets* instead of gamestates as the nodes in the tree. An information set is basically the set containing all indistinguishable gamestates from the the point of view of the player. This allows the sharing of information over different determinizations, which in addition to reducing strategy fusion, provides a basic algorithm with many possibilities for adaptions and improvements. Some of them are implemented in the agents described in the following sections.

4.2 Stupid and Random Agent

To have a baseline, I created the *stupid* and the *random* agent. The *stupid* agent always passes whenever possible and plays a random single card when passing is not allowed. This is the simplest agent able to play the game, but it is also one of the worst possible agents.

The *random* agent always plays an action chosen uniformly at random from the possible actions.

4.3 Minimax Agent

Even though the gametree for a Tichu game is too big for an exhaustive search I implemented an agent using the minimax algorithm (with alpha-beta pruning). To complete one search to a depth of 10 (which corresponds not even to one trick) already takes more than a minute (on my machine). To be able to use it anyway, the agent has the possibility to 'cheat', that is to observe the handcards of the other players, and therefore does not have to search several determinizations.

The simplest *utility* function for the agent is just *14 - "the number of handcards the player has"* since the goal is to get rid of all cards. However this leads the agent always to play the longest combination possible which turns out not to be a good strategy.

run minimax vs random

The fact that creating a good *utility* function needs a lot of *domain knowledge* and that minimax can't deal efficiently enough with the hidden information let me to quickly search for more suitable algorithms.

when doing experiments with utility of minimax, add them here

Side-note: *David Pfander* ([4]) built an agent for Tichu with a carefully crafted *utility* function (requiring a lot of *domain knowledge*) and managed to achieve 'average human level' play.

4.4 Monte Carlo Tree Search Agents

MCTS nicely fits the requirements of "using little *domain knowledge*" and is able to deal with hidden information. That makes it an ideal algorithm for this project, and after reading [6, 7, 8] and their application of ISMCTS on the game *Dou Di Zhu* (which has many similarities to Tichu), I decided to concentrate, for the remainder of the project, on agents using ISMCTS as base algorithm. In particular on following parts of ISMCTS:

- Explore possibilities to make better *determinizations* without using *domain knowledge*
- Reducing or managing the branching factor
- Adding improvements over the ISMCTS and compare them
- Finding a better *default policy*(in the simulation phase) than the random strategy

In the remainder of this section, I discuss different improvements for the ISMCTS-Tichu agents related to one of the 4 parts. Each of those improvements then is evaluated in the chapter Experiments / Tournaments.

4.4.1 Default ISMCTS Agent

The *default ISMCTS agent* builds the basis for the agents improving on the various parts of the algorithm. In this section the *tree policy*, the *state evaluation* and the *selecting of the best action* are determined and used for all other ISMCTS agents, except when stated otherwise.

Tree selection

As hinted at in the *Background* section, the simplest and most often used *tree policy* uses the UCT (Upper Confidence bound for Trees) formula proposed by [13]. It is derived from viewing the

selection problem as a multiarmed bandit problem where each action is an arm and the reward is the result of doing a Monte Carlo simulation after choosing that action.

To be able to use the UCT formula, each node (n) keeps track of the number of times it has been visited (v_n) and the sum of the rewards the player got after visiting the node (r_n).

The *tree policy* is then: For a given node p, select the childnode c that maximizes following formula (ties are broken arbitrarily):

$$UCT = \begin{cases} \frac{r_c}{v_c} + C\sqrt{\frac{2\ln v_p}{v_c}}, & \text{if } v_c > 0 \\ FMU, & \text{otherwise} \end{cases}$$

where r_c is the total reward of the child, v_p is the number of times the current (parent) node has been visited, v_c the number for the child node and $C > 0$ is a constant balancing exploration and exploitation ($\sqrt{2}$ is recommended by [7]).
It is generally recommended that $FMU = \infty$ (First Move Urgency), so that not yet visited child nodes will be visited first before any of them are expanded. However FMU can be set to any value, allowing 'good' children to be expanded before some of their siblings are visited for the first time.

I didn't tune neither C nor FMU and kept them at $\sqrt{2}$ and ∞ respectively.

Evaluation

At the end of a rollout, the final state has to be evaluated and the reward is backpropagated trough the gametree. The evaluation thus plays a central role on determining the UCT value of nodes.

I considered following different evaluation strategies (remember that the players from the same team get the same amount of points at the end of a round):

Absolute Points, Each team gets the unaltered points as reward.

Absolute Normalized Points, The absolute points normalized between -1 and 1

Relative Points, Each team gets the difference of points to the other team as the reward. For example, the round ended 70 : 30, then the first team gets $70 - 30 = 40$ reward, the second $30 - 70 = -40$ reward.

Relative Normalized Points, The relative points normalized between -1 and 1

Ranking Based, This evaluation rewards the teams based on the ranking at the end of the round:

- $+1$ for a doublewin
- -1 for a double loss (enemy has doublewin)
- 0.5 for a player of the team finishing first (but no doublewin)
- 0 when an enemy player finished first (but no doublewin)

It turns out that the *ranking based* evaluation produces better results than any of the other strategies (experiments in section 5.9). Therefore this evaluation was used for all agents.

Selecting the best action at the end

After the search completed, the best action from the initial search-state has to be determined. The recommended strategy is to select the action corresponding to the most visited node. And indeed that is what I ended up implementing after trying out the two other obvious strategies: taking the node with *highest UCB value* and taking the node with *highest average reward*.

The experiments confirming the recommendation can be found in section 5.9.2.

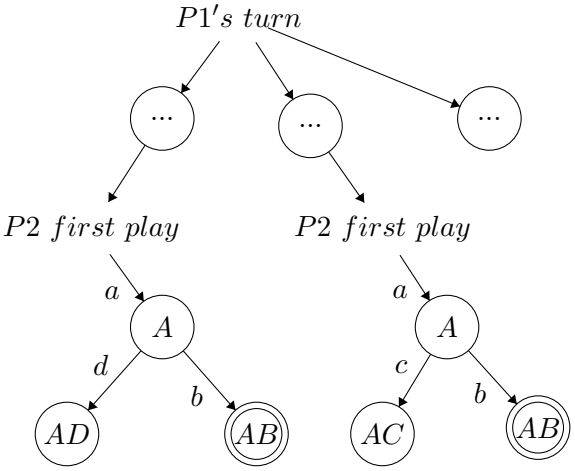
4.4.2 Managing the branching factor

In [7] it is shown that ISMCTS does not perform significantly better than PIMCTS in *Dou Di Zhu*. However, they suggest this is due to the high branching factor of the game and that in each determinization new possible actions are discovered. Reducing those two factors would increase the advantages of ISMCTS over PIMCTS.

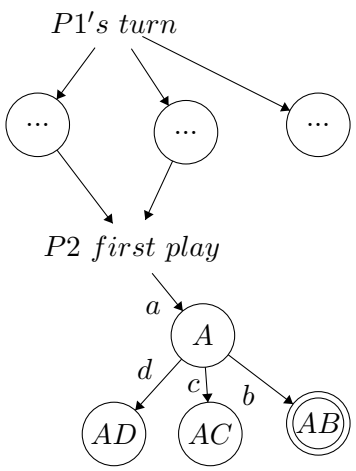
A big reduction of the branching factor comes from the observation that many possible combinations in a given hand are logically identical and thus only one of them has to be considered (see the *Implementation* section).

Other reductions are achieved by the following two enhancements to ISMCTS.

Episodic Information Capture and Reuse (EPIC)



(a) An example of the *position in epic*. The two circled states have the same *position in epic* but are in a different part of the gametree. Both are reached by action a followed by b from a state where player 2 can play first. However state AD and AC have different *position in epic*. The nodes labeled with A and *P2 first play* also have the same *position in epic*



(b) The EPIC-gamegraph corresponding to (a). The two nodes labeled AB are merged as well as the states labeled A and the state where Player 2 can play first.

Figure 4.2: EPIC example

EPIC is an enhancement proposed by [8] and aims to take advantage of the episodic nature of certain games. It yielded good improvements in *Dou Di Zhu* and it stands to reason that it does also help in Tichu. *Dou Di Zhu* has the same trick system as Tichu, but it is played with only 3

players and the playable combinations are slightly different. But overall the two games are very close.

In case of *Dou Di Zhu* and Tichu, an episode is defined as an entire trick (from first play until all players pass consecutively). The *position in episode* of a node is the path from the node upwards until the beginning of the trick is reached. An example is shown in Fig. 4.2a EPIC assumes that two different nodes that have the same *position in epic* are correlated. Thus, all states with the same *position in epic* are merged into one node creating an EPIC-gamegraph. The result is shown in Fig. 4.2b. The MCTS is then performed on this gamegraph. The idea is instead of

In *Dou Di Zhu* as well as in Tichu one trick

Instead of searching the traditional gametree, EPIC searches an "episode graph" in which each node EPIC assigns each node a "position in episode" In [8] they defined one trick as an episode EPIC changes how the gametree is structured. Basically 4 trees (one per player). The root is the set of all game-states where the player can play first.

Move groups

improvement described in [9]. The idea is to group actions into "move groups" and then in tree selection first select a group (with ucb) and then an action in that group (again with ucb). - Reduces branching factor - sometimes helps to select better actions with fewer iterations - not trivial to find good groups (I just put the same combination-type into the same groups with the idea that they most of the time lead to similar results.)

4.4.3 Determinization

- determinization takes a gamestate with hidden information and creates a perfect information gamestate that is consitent with the 'hidden info' gamestate. in [8, p. 54+] it is shown that for the game *Dou Di Zhu*, more than 20 determinizations per search yield diminishing returns, and the more simulations are made, the better the agent plays.

Random

- just randomly assigns the unknown cards to the other players.

Probabilities

- Based on some prior probability collected from real game data. - how data is collected - introduce "General Combinations" - prior for all nbr handcards (1-14) and for all gencombs: nbr of handcards -¿ proba of containing general combination.

Data analysis

- look at the collected data - distribution of single cards and pairs are significant different from uniform random

Pool det

- a determinization strategy distributing whole combinations on the other players based on 'which combination is most probable to be present and who is most probable to have it'

Single det

- looks only at each player in turn and only at single cards. - "given a player, which cards are most probable to be in his handcards"

NN det

- experimental - tried to learn for each nbr of handcards (N) one model 'given all (unknown) cards, which N cards are most probable to appear' - did not converge. probably data is too noisy and "near misses" are not counted as correct. There is no concept of 'closeness' of the cards. (ie. giving enemy a 10 instead of a 9 is less 'wrong' than giving a K)

4.4.4 Rollout

The rollout phase simulates from a given gamestate to a terminal one following a (fast) action selection strategy. The goal is to estimate an initial value for the state.

random

- selects always a random (legal) action. - Fast but only good when enemy plays randomly.

Last Good Response With Forgetting (LGRF)

- still random action selection, but stores all selected action pairs that lead to a win. When encoutering a stored action, then the response (second action in the pair) is chosen instead of random action. - forgetting is an additional feature, where all stored action responses that lead to a loss are deleted (forgotten).

no rollout

- There is the possibility of no rollout. That is all actions are added to the search tree and chosen according to ucb. this is still a random rollowut for states that have never been visited before.

dqn-nn

- Instead of choosing the next action randomly, a NN agent (see 4.5) chooses the next action. - Used in AlphaGo

4.5 NN Agents

Implementation / Keras-rl - several network architectures - several training environments (enemies)

4.5.1 Architectures

- cards as 1-hot encoded vector - cards grouped by rank vector - seperate inputs - current ranking input

4.5.2 Training

- against random - against former version of itself - against itself - against (fast) mcts

- policy (Bolzman with variable or fixed tau)

Experiments / Tournaments

- Tournament of some agents: A team for each agent (containing 2 identical agents), Each team then plays against every other team once. depending on how close the agents are to each other and how long one game takes to simulate, a game may go form 1'000 points up to 100'000 points. The difference of the points is then the measure on how much better one agent is to another. there is of course a variance (depending mostly on the dealt cards and luck). This is why sometimes more and longer games were simulated.

5.1 minimax, random, mcts tournament

5.2 cheat vs noncheat

- cheat (not surprisingly) better

5.3 reward tournament

- reward based on ranking is best. - absolute and relative reward suffer from the fact that points reach from -200 to 200 but most of the time they are between 0 an 100. also suffer from the problem that when an agent could finish (which is practically always the best action), he does not because reward depends also on how the teammate plays. and with random rollout may vary a lot from simulation to simulation (agent can't distinguish between his own and teammate actions) - bad luck in rollout may give bad score to a good action -¿ -200 is too small

5.4 random vs pool vs single determinization

5.5 rollout tournament

5.6 split tournament

5.7 dqn tournament

5.8 best vs best without some enhancement

5.9 state evaluation

5.9.1 discussion of results

Note that the evaluations suffer from the *credit- assignment problem* (both agents get the same reward, regardless of the order they finished) but they encourage teamplay.

5.9.2 best action

5.10 against human / overall playingstrength

Summary

6.1 Results

6.2 Future work

6.3 Lessons Learned

maybe could have found framework (eg. tichumania, or other sites) Implementing took most of the time.

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Author's Decleration

s

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Appendix

Bibliography

Bibliography

[1] Fata Morgana: http://www.fatamorgana.ch/tichu/tichu.asp

[2] Tichu Rules and strategies: http://scv.bu.edu/∼aarondf/Games/Tichu/

[3] Jeffrey Long, Nathan R. Sturtevant, Michael Buro, Timothy Furtak, **Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search**, Department of Computing Science, University of Alberta Edmonton, Alberta, Canada in Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010

[4] David Pfander, **Eine künstliche Intelligenz für das Kartenspiel Tichu**, Universität Stuttgart, 2013.

[5] Stuart Russel, Peter Norvig **Artificial Intelligence, a modern approach**, third edition, 2014.

[6] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis and Simon Colton, **A Survey of Monte Carlo Tree Search Methods**, IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 4, NO. 1, MARCH 2012

[7] Peter I. Cowling, Edward J. Powley, Daniel Whitehouse, **Information Set Monte Carlo Tree Search**, IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 4, NO. 2, JUNE 2012

[8] Daniel Whitehouse, **Monte Carlo Tree Search for games with Hidden Information and Uncertainty**, University of York, August 2014

[9] Benjamin E. Childs, James H. Brodeur, **Transpositions and Move Groups in Monte Carlo Tree Search**,

[10] Gabriel Van Eyck, Martin Müller, **Revisiting Move Groups in Monte Carlo Tree Search**, University of Alberta, Edmonton, Canada, January 2012

[11] D. Robilliard, C. Fonlupt, and F. Teytaud **Monte-Carlo Tree Search for the Game of "7 Wonders"**, LISIC, ULCO, Univ Lille–Nord de France, FRANCE

[12] Diego Perez, Spyridon Samotharakis, Simon M.Lucas, **Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games**, Juli 2013

[13] L. Kocsis and C. Szepesvári, **Bandit based Monte-Carlo Planning** in Euro. Conf. Mach. Learn. Berlin, Germany: Springer, 2006, pp. 282–293.

[14] I. Frank and D. Basin, **Search in games with incomplete information: A case study using Bridge card play** Artif. Intell., vol. 100, no. 1–2, pp. 87–123, 1998.