Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Katja Tuma

Drevesno preiskovanje Monte Carlo s Thompsonovim vzorčenjem pri igri Prebivalci otoka Catan

magistrsko delo

študijski program druge stopnje računalništvo in informatika

Ljubljana, 2016

University of Ljubljana
Faculty of Computer and Information Science

Katja Tuma

Monte Carlo Tree Search with Thompson Sampling in the Settlers of Catan

master's thesis

master's study program
computer science

Mentor: Prof. Dr. Branko Šter
Comentor: Prof. Dr. Bengt J. Nilsson

Ljubljana, 2016

Povzetek

Naslov: Drevesno preiskovanje Monte Carlo s Thompsonovim vzorčenjem pri igri Prebivalci otoka Catan

Drevesno preiskovanje Monte Carlo (MCTS) je ena izmed najbolj uporabljenih metod pri implementaciji močnega računalniškega igralca iger v umetni inteligenci, brez uporabe predhodnega znanja o domeni. Najmočnejši in najbolj popularni algoritmi, ki se pogosto uporabljajo za rešitev t.i. dileme *raziskovanja (engl. exploration)* proti *izkoriščanju znanja (engl. exploitation)* pri problemu več-rokih banditov, so raziskani in predstavljeni s pomočjo pregleda literature. Na podlagi empiričnih študij Thompsonovega vzorčenja v primerjavi s pristopom zgornje meje zaupanja (UCB) ter različicami podobnih algoritmov smo v magistrskem delu spremenili drevesno strategijo širjenja v MCTS. Končna domena aplikacije spremenjenega algoritma je družabna igra Prebivalci otoka Catan (SoC), implementirana v programskem jeziku C, skupaj z MCTS-UCT agentom, MCTS-TS agentom ter dvema preprosto igrajočima agentoma. Meritve učinkovitosti naštetih agentov prikazujejo povečano moč igranja agenta s spremenjeno drevesno strategijo, v primerjavi z najbolj pogosto uporabljenim pristopom, t.j. UCT.

Keywords: drevesno preiskovanje Monte Carlo (MCTS), več-roki bandit (MAB), zgornja meja zaupanja pri drevesih (UCT), Thompsonovo vzorčenje (TS), umetna inteligenca (AI), Prebivalci otoka Catan (SoC).

Abstract

Title: Monte Carlo Tree Search with Thompson sampling in The Settlers of Catan

Monte Carlo Tree search (MCTS) is a popular method of choice for addressing the problem of a strong computer based game playing agent in Artificial Intelligence, without any prior domain knowledge. The strongest and most popular algorithms used to tackle the so-called *exploration* vs. *exploitation* dilemma in Multi-armed Bandit (MAB) problems were identified and presented in a literature review. Empirical studies measuring the performance of Thompson sampling (TS) and the state-of-the-art Upper Confidence Bound (UCB) approach in the classical MAB problem have been found, results of which support our modified tree policy in MCTS. The domain of application is the board game of the Settlers of Catan (SoC), which is implemented as a multi-agent environment in the programming language C, along with a MCTS-UCT agent, MCTS-TS agent and two strategy playing agents, namely the ore-grain and wood-clay agent. Performance measurements of the aforementioned agents, presented and discussed in this work, demonstrate an increase in the performance of the agent with the modified tree policy, when compared to the state-of-the-art approach (UCT).

Key words: Monte Carlo Tree Search (MCTS), Multi-armed Bandits (MAB), Upper Confidence Bound for Trees (UCT), Thompson sampling (TS), Artificial Intelligence (AI), the Settlers of Catan (SoC).

I would first like to thank my thesis advisor and mentor Prof. Dr. Branko Šter of the Faculty of Computer and Information Science in Ljubljana for guiding me through every obstacle I had to overcome while creating this work. The door to Prof. Dr. Branko Šter was always open for whenever I ran into trouble or needed advice.

I would also like to thank my co mentor Prof. Dr. Bengt J Nilsson of the Malmö University for helping me with the research process, steering me in the right direction and for providing valuable input about this thesis for which I am most grateful.

Finally, I must express my gratitude to my close friends and family for unconditionally providing me with support and motivation I needed at every step of

the way. This accomplishment would surely not have been possible without them. Thank you.

Katja Tuma.

Table of Contents

Table of Abbreviations

abbreviation	slovene	english
AI	Umetna inteligenca	Artificial Intelligence
AI-based	Osnovano na umetni inteligenci	Artificial Intelligence based
SoC	Prebivalci otoka Katan	Settlers of Catan
SOS	Vsota polj	Sum of Switches
MAB	Več-roki bandit	Multi-armed Bandit
UCB	Zgornja meja zaupanja	Upper Confidence Bound
TS	Thompsonovo vzorčenje	Thompson sampling
SDP	Stohastično dinamično programiranje	Stochastic Dynamic Programming
MCTS	Drevesno preiskovanje Monte Carla	Monte Carlo Tree Search
UCT	Zgornja meja zaupanja pri drevesih	Upper Confidence Bound for Trees
RAVE	Hitra ocena akcij	Rapid Action Value Estimation
AMAF	Princip prvič obiskanih potez	All Moves As First principle
MC-RAVE	Hitra ocena akcij po Monte Carlu	Monte Carlo Rapid Action Value Estimation
UCT-RAVE	Hitra ocena akcij po zgornji meji zaupanja pri drevesih	Upper Confidence Bound for Trees Rapid Action Value Estimation
MDP	Markovski odločitveni proces	Markov Decision Process
GUI	Uporabniški grafični vmesnik	Graphical User Interface
PDF	Gostota verjetnosti	Probability density function

Razširjen povzetek

To poglavje vsebuje kratek opis celotne vsebine magistrskega dela. Razširjen povzetek opisuje poglavje o domeni aplikacije, tj. poglavje o igri Prebivalci otoka Catan, poglavje o problemu več-rokega bandita, poglavje o drevesnem preiskovanju Monte Carlo ter poglavje o Thompsonovem vzorčenju v drevesnem preiskovanju Monte Carlo. Opis slednjega poglavja vsebuje tudi kratko diskusijo rezultatov meritev, pridobljenih v fazi testiranja novega algoritma.

Igra Prebivalci otoka Catan (SoC) je družabna igra, prvič predstavljena trgu leta 1995 pod avtorstvom Klausa Teuberja. Pravila igre se razlikujejo od posamezne različice, saj so se od nastanka osnovne igre pojavile razširitve, tako števila možnih igralcev, kot poteka igre. Za namene magistrske naloge je bila izbrana prvotna različica igre, kjer sodelujejo štirje igralci. Potek igre je v postavitveni fazi nekoliko drugačen kot v igralni fazi. V postavitveni fazi je potrebna postavitev igralne plošče ter postavitev začetnih naselij. V igralni fazi pa se igra odvija tako, da igralci izmenično mečejo kocki ter uporabljajo svoje resurse za gradnjo oziroma razvoj naselij. Izid posameznega meta kock povzroči produkcijo resursov tistim igralcem, ki imajo svoja naselja postavljena okoli aktivirane šestkotne plošče. Trenutni igralec ima možnost izbire ene ali več sledečih akcij:

1. igralec lahko izvede menjavo kart z ostalimi igralci ali z banko,
2. igralec lahko gradi naselje, mesto ali cesto,
3. igralec lahko kupi ali igra predhodno kupljeno karto za razvoj ter
4. igralec lahko prepusti igro naslednjemu igralcu.

Posamezna naselja ter razvojni cilji so nagrajeni z določenim številom točk. Igra se konča, ko eden izmed igralcev prvič doseže 10 točk.

Problem več-rokega bandita (MAB) je bil prvič omenjen leta 1952 ter je v literaturi pogosto opisan na primeru igralca, ki igra na igralnem avtomatu v igralnici. Cilj igralca je igrati tako zaporedje ročic, do bo skupen seštevek dobitkov največji. Za reševanje problema več-rokega bandita se pogosto uporablja izračun Gittinsovega kazalca ter izračun vrednosti zgornje meje zaupanja (UCB). Poleg omenjenih pristopov, smo se odločili raziskati tudi učinkovitost implementacije Thompsonovega vzorčenja (TS) pri problemu več-rokega bandita.

Drevesno preiskovanje Monte Carlo (MCTS) je algoritem, ki preiskuje prostor drevesne strukture na osnovi naključnih simulacij. V osnovi gre za algoritem, ki vsebuje *drevesno strategijo* za izbiro najboljše poti do lista drevesa ter *privzeto strategijo* za določanje rezultata naključnih simulacij. Bolj podrobno, drevesno preiskovanje Monte Carlo vsebuje štiri korake, ki se izvedejo ob vsaki iteraciji algoritma:

1. izbor lista drevesa v skladu z drevesno strategijo,
2. razširitev drevesa z novim vozliščem,
3. naključna simulacija v skladu s privzeto strategijo ter
4. posodobitev vozlišč drevesa na poti od novega lista do korena.

Pri drevesnem preiskovanju je razširitveni faktor (tj. število možnih potez) bistvenega pomena. Posledično je potrebna uporaba učinkovitega načina ocenjevanja posameznih vozlišč. Preprosta ali uniformna rešitev za spopadanje s t.i. dilemo *raziskovanja* proti *izkoriščanju znanja* v literaturi ni bila zasledena. V magistrskem delu so raziskani najbolj pogosto uporabljeni pristopi, kot je zgornja meja zaupanja pri drevesih (UCT), hitra ocena akcij (RAVE) ter uporaba hevristike.

Pred implementacijo Thompsonovega vzorčenja v drevesno strategijo drevesnega preiskovanja Monte Carlo, je bila raziskana učinkovitost uporabe Thompsonovega vzorčenja v primerjavi z uporabo izračuna zgornje meje zaupanja pri enostavnem problemu več-rokega bandita. Opravljene meritve obžalovanja, prikazane na Sliki 2, prikazujejo počasnejše naraščanje obžalovanja izbire ročic pri uporabi Thompsonovega vzorčenja.

Naš program je implementiran v programskem jeziku C in sestoji iz MCTS-TS agenta, MCTS-UCT agenta ter dveh preprosto igrajočih agentov. Testiranje delovanja MCTS-TS agenta je bilo izvedeno na dva načina:

1. testiranje igranja proti človeškemu igralcu (avtorju magistrskega dela) s pomočjo preprostega grafičnega vmesnika ter

2. merjenje povprečnega razmerja zmag in obžalovanja posameznega agenta pri različnem številu simulacij na MCTS potezo.

Postopoma so bile razvite in testirane tri različice programa. Prvotna različica programa vsebuje logiko igre, pri kateri je možnih $2^6 = 64$ potez. Vsaka poteza je kombinacija naslednjih akcij:

1. izmenjava kart z banko,

2. gradnja ceste,

3. gradnja naselja,

4. gradnja mesta,

5. nakup karte za razvoj ter

6. igranje karte za razvoj.

Meritve prvotne različice programa, predstavljene na slikah 9 in 10, prikazujejo boljše delovanje MCTS-TS ter MCTS-UCT agenta v primerjavi z agenti, ki igrajo z upoštevanjem preproste strategije. Kljub vsemu pa je razvidno, da je MCTS-TS agent v večini primerov premagan s strani MCTS-UCT agenta. Zaradi visokega razširitvenega faktorja je bila razvita druga različica programa, kjer je logika igre spremenjena tako, da je možnih le 7 akcij na potezo [1]. Rezultati, pridobljeni z drugo različico programa ter prikazani na slikah 11 in 12, niso uspeli pokazati boljšega delovanja MCTS-TS agenta v primerjavi s prvotnim delovanjem. Pravzaprav je bilo med testiranjem opaženo, da se igra odvija počasneje, saj so spremembe logike igre bistveno spremenile potek igre, zaradi česar je bila tretja verzija programa razvita na osnovi prvotno implementirane igre. Pri tretji verziji programa je uporabljeno t.i. *posteriorno preoblikovanje* Beta porazdelitve, s pomočjo katere poteka Thompsonovo vzorčenje. Meritve, predstavljene na slikah 7 in 8, prikazujejo hitrejšo rast obžalovanja MCTS-UCT agenta v primerjavi z

[1] Poleg prej naštetih akcij je možna tudi predaja igre naslednjemu igralcu.

MCTS-TS ter višje vrednosti povprečij zmag MCTS-TS v primerjavi z MCTS-UCT. Vpeljavo Thompsonovega vzorčenja v drevesno strategijo algoritma MCTS in implementacijo le-tega v domeno Prebivalci otoka Catan predstavljamo kot naš glavni prispevek magistrskega dela.

Chapter 1

Introduction

Game theory has for decades played an important role in advances made in various fields, such as computer science, economics, political science, biology and many more. In fact, the earliest example of a formal game-theoretic analysis was an economical-oriented study of duopoly by Antoine Cournot in 1838. Game theory is a formal study of mathematical models of conflict and cooperation between intelligent rational decision-making agents. It has especially received attention after the introduction of the *Nash equilibrium* in 1950. With it, John Nash demonstrated that finite games have always an equilibrium state, at which all players choose their best action according to the opponents' choices. Having said that, the mathematical principles defined in game theory have been applied on a variety of problems where an automated decision-making system that interacts with the environment is needed. Our work is focused particularly on a non-zero sum game with non-cooperative competing decision-making agents.

The main objectives of the thesis are to identify a novel approach of a MCTS-based solution for a game playing agent of the board game the Settlers of Catan, implement the game logic along with the state-of-the-art solution and the novel approach in the programming language C, and finally to obtain the performance feedback from measuring regret and average winning rate of several agents.

Accordingly, the Master's Thesis is comprised of multiple chapters and sections. In the second chapter, we introduce the reader to the board game and the rules of playing the game. We further elaborate on some of the rule changes made, in order to simplify the game logic for faster computation. We finish the chapter

with a section of previous implementations of Catan playing agents. In the third chapter we continue with presenting the classical MAB problem along with the most frequently proposed solutions. Similarly, the fourth chapter describes in detail the Monte Carlo Tree Search with its popular varieties and enhancements. As the classical MAB problem and MCTS tackle with a similar dilemma, notably the exploration vs. exploitation dilemma, the fifth chapter finally discusses our method of choice in solving the problem at hand. Our results are presented within the same chapter along with the discussion.

Chapter 2

The Settlers of Catan

Settlers of Catan (SoC) is a non-deterministic from two to six player board game, designed by Klaus Teuber and first published in 1995 in Germany under the name *Die Siedler von Catan*. The game has since been one of the most popular strategic board games and has gained popularity on a global scale, selling more than 22 million copies in 30 different languages around the world. The players assume the roles of conquerors, seeking to build and develop their initial settlements through acquiring and trading various resources, for which they are awarded points. Successful game plays lead towards eventually reaching 10 points before the competing players and consequently winning the game. Even though the standard number of players is set to be from three to four, the game has been developed to include expansions where additional rules are applied. For the purposes of our research, we will further elaborate on game rules of the standard four player set-up. We will proceed to argue the rule changes that were necessary for a simpler implementation of the game logic and finish the chapter with presenting some previous work done in this research area, also mentioning the most important Artificial Intelligence based (AI-based) game implementations available today.

2.1 Game rules

The standard set-up of the SoC consists of four players and several game components:

(a) 18 resource terrain tiles and 1 dessert tile,

(b) 6 sea frame pieces that together contain 9 ports,

(c) 18 number tokens and 1 robber,

(d) 95 Resource Cards of clay, ore, sheep, wheat and wood,

(e) 25 Development Cards (14 Knight Cards, 6 Progress Cards and 5 Victory Point Cards),

(f) 2 Special Cards: Longest Road and Largest Army,

(g) 16 city pieces,

(h) 20 settlement pieces,

(i) 60 road pieces,

(j) 2 dice.

The game is played in two phases: the initial set-up phase and the game phase. The initial phase comprises of island construction, distribution of pieces and positioning two settlement and road pieces of each player on the board. There are various possibilities for island construction, in fact, a random approach can be followed if desired. However, certain pre-set combinations of resource tiles and production numbers are proposed by the accompanying booklet of rules. Each player selects a color and collects the corresponding 5 settlements, 4 cities and 15 roads and 1 Building Cost Card. Players follow a simple algorithm for determining the order of first settlement and road positioning. All players roll the dice once, and remember their outcome sum. In the first round of initial placements a descending order of players' outcomes is applied whereas in the second, an ascending order is applied. This procedure balances out the advantage of selecting from an empty board and having the opportunity to play both phases at once. After the initial phase the game may begin. In each turn the player first rolls the dice. The production number matching the outcome activates the distribution of the corresponding resources to the players that positioned their settlements on the adjacent intersections of the tile. After the Resource Cards are obtained, the player in turn is allowed to make several moves before passing the turn to other players. The

player is able to trade resources, build roads, settlements and cities, buy and play Development Cards. The following paragraphs describe the rules of each move in detail.

2.1.1 Resource production

Each player who has a settlement on an intersection marked with the number corresponding to the outcome of a dice roll, receives one Resource Card of the tile's type. If the player has two or three settlements bordering the same tile, he/she receives one Resource Card for each settlement. The players receive two Resource Cards of the same type for each city bordering the active tile. If there is not enough Recourse Cards in the main card deck to supply everyone, no one receives any resources that turn. If a player rolls a 7, instead of resource distribution, the following happens before the trading can continue:

(a) All players that posses more than seven resources must select half (rounded down) and return it to the supply stack,

(b) The player in turn has to move the robber to another terrain tile to block resource production,

(c) The player can choose to steal one Resource Card from any player occupying the selected terrain tile. [1]

2.1.2 Trade

After the Resource Production each player is allowed to trade freely, using one or both types of trading. In a Domestic Trade, the player in turn is allowed to announce what type of resource he wants to trade for what price. Other players are allowed to trade only their Resource Cards with the player in turn, while trying to negotiate for the best possible trade. In a Maritime Trade, the player in turn is allowed to trade resources with the supply stack, exchanging multiple cards of the

[1] The robber deactivates the terrain tile. All players that have settlements or cities positioned around the blocked tile do not get any resources when the production number is rolled. The robber may be moved to the desert tile, where it does not block any production until the next seven is rolled or a Knight Card is played.

same resource for one desired Recourse Card. If the player possesses a settlement or a city on a port intersection, the trading price is 2:1 (two Resource Cards of the port's type for one desired Resource Card). There are four out of nine ports that support trading of all resources for the price 3:1. If the player in turn is not occupying any ports, the price for Maritime Trade is 4:1.

2.1.3 Build

Once the trading has been finished, the player in turn can proceed to build new elements on board to gain Victory Points, expand the territory, improve the resource production and/or buy Development Cards. In order to build the players must pay a specific combination of resources to the supply stack.

(a) Road: 1 clay & 1 wood
A new road must always connect to one of the player's existing roads, settlements or cities. Only one road can be build on a given path. The first player to build a continuous road of at least 5 road pieces (not counting forks), obtains the Longest Road Card, which is worth two Victory Points. If another player exceeds the current longest road, the Longest Road Card is stolen, along with the corresponding points.

(b) Settlement: 1 clay & 1 wood & 1 sheep & 1 wheat
A settlement can only be build at an intersection if all three adjacent intersections are unoccupied and must be at least connected to one road. Each settlement is worth one Victory Point and results in one additional Resource Card when active after the dice roll.

(c) City: 3 ore & 2 wheat
A city can only be build upon a previously built settlement, where the settlement piece is taken off the board for further use, as the game unfolds. Each city is worth two Victory Points, and results in two additional Resource Cards when active after the dice roll.

(d) Buying a Development Card: 1 ore & 1 sheep & 1 wheat
There are three types of Development Cards: Knight, Progress Card and

Victory Point. After this card is bought and played it is never returned to the supply stack, as it remains in the hands of the player.

2.1.4 Playing Development Cards

At any time during the turn, the player is allowed to play one Development Card bought in one of the previous rounds. The Knight Card activates the robber, therefore the same procedure is followed, as when the dice roll outcome equals to seven. When a player has collected and played three Knight Cards, he/she obtains the Largest Army Card, which is worth two Victory Points. If another player exceeds the number of played Knight Cards, the Largest Army Card is stolen, along with the corresponding points. There are three types of Progress Cards: Road Building, Year of Plenty and Monopoly. If the player chooses to play the Road Building Card, he/she can immediately build two roads for free. If the Year of Plenty Card is played, the player can choose two Resource Cards from the supply stack. When the Monopoly Card is played, the player chooses one type of resource and steals all resources of the same type from other players. A Victory Point Card remains hidden until the last move, when the player in turn is sure to have all 10 points.

2.2 Rule changes

From the rules of the SoC, it is apparent that the complexity of game logic even increases with using extensions, so our first limitation was to implement the game with a standard set-up model of four players. Following the example of rule changes introduced by Szita et al. in [17], we further limited our agents to Maritime Trade only. Similarly to the opinion of aforementioned authors, we believe that these rule changes do not significantly alter the game, nevertheless, they do handicap our agents playing strength. In addition, for the purpose of reducing the measuring time of several game simulations, the end condition was changed (only 7 Victory Points required for victory).

2.3 Previous implementations

There have been several attempts in developing strong playing agents in the SoC, including the use of Reinforcement learning strategies by Pfeiffer in [15] that some consider to be the first step in applying advanced machine learning techniques for solving complex game problems. Moreover, researchers have also been exploring the validity of using multi-agent approaches to create game playing bots with a centralized logic, as is demonstrated by Branca and J. Johansson in [4]. Classic approaches in AI require either a high level of domain knowledge or a long response time from game playing agents in complex game situations. Monte Carlo Tree Search (MCTS), on the other hand, requires very little or no domain knowledge, using only randomized simulations with a pre-defined tree policy. Indeed, the implementation of MCTS based playing agents has been repeatedly applied to a great variety of deterministic two-player games, such as game of go, chess, checkers, solitaire, travelling salesman problem and more, as outlined in the survey of MCTS methods by Browne et al. [5]. Moreover, MCTS has also been applied in non-deterministic multi-player games, in particular Szita et al. [17], have shown that MCTS can be adapted successfully to the SoC board game. Their implementation of the game, *SmartSettlers* is a Java based program, using the open-source client-server oriented implementation, *JSettlers*, as the baseline with GUI. Szita et al. [17] mention other computer implementations of the game, namely *Castle Hill Studios's* version, part of *Microsoft's MSN Games*.

Note that some other work which does not focus on AI agent development specifically, is nevertheless closely related to the game, and can therefore provide a good understanding of the game. Furthermore, when implementing Monte Carlo Tree Search in the SoC, there is a level of abstraction needed for a well planned implementation of agents. Such a framework is proposed by G.J.B. Roelofs in [16].

Chapter 3

Multi-armed Bandit problem

First introduced by Herbert Robbins in 1952, a Multi-armed Bandit (MAB) problem can be regarded as the problem in which a gambler is playing a set of slot machines (sometimes referred to as arms) at a Casino. The gambler's objective is to play the best arm sequence, in order to maximize the sum of rewards. Essentially, the algorithms designed to handle a MAB problem, typically find balance between the so called *exploration* and *exploitation*, iteratively optimizing and guiding the gambler to find the best arm. Such problems arise on many occasions, notably in the context of on-line planning, ad placement in web advertisement, clinical trials, etc.

There are at least as many approaches to tackle the exploration vs. exploitation dilemma, as there are variants of MAB problems. The survey on MAB problems performed back in 2008 by Mahajan and Teneketzis [13] mentions the following: Superprocesses, Arm-acquiring Bandits, Switching Penalties, Multiple Plays and Restless Bandits. Several of these are related with one another and can be sometimes converted into another. Furthermore, there are optimal and approximate strategies, depending on how much accuracy can be sacrificed for efficiency. To only name a few: semi-uniforms, Thompson sampling, Pricing poker, LinUCB, Kernel UCB, Gittins index, etc. Moreover, studies have been conducted, researching the empirical evaluation and effectiveness of several theoretically well-understood approaches. One of such studies was performed by Kuleshov and Precup [11], where the authors empirically study the most popular solutions to MAB problems, namely ϵ-greedy, Boltzmann Exploration (Softmax), Pursuit Al-

gorithms, Reinforcement Comparison and Upper Confidence Bound (UCB), in the context of clinical trials. In this chapter we seek to establish a formal description of a Classical MAB problem and present ways for solving such problems.

3.1 The Classical MAB

According to Mahajan and Teneketzis in [13], MAB problems are a class of sequential resource allocation problems concerned with allocating one or more resources among several alternative (competing) projects. A *bandit process* is defined as a special type of Markov Decision Process in which there are two possible actions: freeze and continue. The latter produces a reward and results in a change of state according to Markov dynamics. More specifically, the classical MAB problem is a collection of k independent single-armed bandit processes. It therefore also consists of the so called controller or processor. At each step the controller chooses to operate exactly one arm while the others remain frozen. In order to demonstrate how the system evolves, Mahajan and Teneketzis [13] assume that each arm i, $i = 1, 2, ..., k$ is represented by sequences $(X_i(N_i(t)), R_i(X_i(N_i(t))))$; $N_i(t) = 0, 1, 2, ..., t$; $t = 0, 1, 2, ...$, where N_i denotes the number of times the arm has been pulled until time t and R_i denotes the reward generated by arm i at time t. They further assume that $U(t) = (U_1(t), ..., U_k(t))$ denotes the action taken by the controller at time t and $W_i(n)$; $i = 1, ..., k$; $n = 0, 1, ...$ a sequence of state-independent variables. The system evolves according to

$$X_i(N_i(t)) = \begin{cases} X_i(N_i(t)), & \text{if } U_i(t) = 0, \\ f_{N(t)}(X_i(0), ..., X_i(N_i(t)), W_i(N_i(t))), & \text{if } U_i(t) = 1, \end{cases} \tag{3.1}$$

and

$$N_i(t+1) = \begin{cases} N_i(t), & \text{if } U_i(t) = 0, \\ N_i(t) + 1, & \text{if } U_i(t) = 1, \end{cases} \tag{3.2}$$

for all $i = 1, 2, ..., k$, therefore t represents the local time of each arm, only increasing when $U_i(t) = 1$. The MAB problem, originally formulated in 1940 determines the so called *scheduling policy* that maximizes the accumulated reward of pulling arms until time $t = k$. The problem was known to be solved using Stochastic Dynamic Programming (SDP) techniques, despite its unoptimized approach. Other

possible approaches to solving the problem evolved, one of them discovered by Gittins and Jones in their work dating back to 1972 [9]. In the following sections we briefly describe the idea behind Gittins index and continue to show the possible application of TS in MAB.

3.2 Gittins index

The Gittins index is a measure of reward (a real scalar value) associated to the state of a stochastic process with a reward function and a probability of termination. The arm is chosen based on the Gittins index. Such a policy of choice is commonly referred to in the literature as the index policy and it follows the Theorem 3.2.1, originally proved to be the optimal solution by Gittins and Jones and others (eg. short proof by Tsitsiklis [18] and alternative proof by Weber [19]).

Theorem 3.2.1 *(Gittins index Theorem) The expected discounted reward obtained from a simple family of alternative bandit processes is maximized by always continuing the bandit having greatest Gittins index*

$$G_i(x_i) = \sup_{r \geq 1} \frac{E\left[\sum_{t=0}^{\tau-1} r_i(x_i(t))\beta^t \middle| x_i(0) = x_i \right]}{E\left[\sum_{t=0}^{\tau-1} \beta^t \middle| x_i(0) = x_i \right]},$$

where τ is a stopping time.

Since the algorithm's discovery, many variations have been applied on several types of MAB problems; nevertheless, there are certain difficulties with Gittins index that we wish to mention. The first one is, that it is very hard to compute, especially where performance is of utmost importance. Another problem is the required independence of arms, which results in an unknown performance and optimality in some applications.

3.3 Upper Confidence Bound

For MAB problems it is useful to determine the Upper Confidence Bound (UCB) that a certain arm will be the optimal choice. The UCB class of algorithms have

been introduced by Lai and Robbins in [12] where they show that such algorithms guarantee that the number of inferior arms pulled is bounded. Furthermore, Auer et al. have proposed a simple version of the UCB algorithm, named UCB1 in their Finite-time analysis of the MAB problem [2]. The proposed strategy focuses on pulling an arm with the maximum value of

$$UCB1 = \bar{X}_j + \sqrt{\frac{2\log n}{n_j}}, \tag{3.3}$$

where $\bar{X}_j$ denotes the average reward from arm j, n_j the number of times arm j has been pulled and n the overall number of pulls so far. Note that the first part of the sum encourages the *exploitation*, while the second part encourages the *exploration* of less visited arms. There have been several developments and variations of the UCB algorithm, some of which are mentioned in the survey by Browne at al [5]. This approach can be directly implemented into trees which we further discuss in Chapter 4 in section Upper Confidence Bound for Trees (UCT).

3.4 Thompson sampling

Dating back to 1933, Thompson studied the problem of finding out which one of two drugs was better when testing them on a patient population under the constraint that as few people as possible should be subjected to the inferior drug. He suggested to adjust the proportions of the future test subjects to the probabilities P and $P - 1$. This ensures that the future test subjects are more often a part of superior treatment, rather than inferior one.

Mellor defines Thompson sampling (TS) in his dissertation [14] as a randomised probability matching (also referred to as posterior sampling) strategy for the MAB problem. For each decision, the probability of an arm being pulled matches the probability that the arm is in fact the optimal arm, given all past observations of arm pulls.

Assuming the regular contextual bandit settings, this paragraph provides a formal description of TS in MAB. At each round the algorithm is able to choose an action a from the set of actions $\mathcal{A}$ and observe the reward r. TS can be best explained by establishing a set of past observations $\mathcal{D}$ that contains pairs of actions and rewards (a_i, r_i), modelled by a parametric likelihood function $\mathcal{P}(r|a, \theta)$

depending on parameters θ. Given some prior distribution $P(\theta)$, the goal is to select actions such as to maximize the expected reward, $\max_a \mathbb{E}(r|a, \theta^*)$. For a simple demonstration, Algorithm 1 outlines the procedure of TS.

Algorithm 1 Thompson sampling

$D \leftarrow \emptyset$
for $t = 1, ..., T$ **do**
  Draw θ^t according to $\mathcal{P}(\theta|\mathcal{D})$
  Select action $a_t = \text{argmax}_a \mathbb{E}_r(r|a, \theta^t)$
  Observe reward r_t
  $\mathcal{D} = \mathcal{D} \cup (a_t, r_t)$
end for

Chapter 4

Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a simulation based search algorithm. It has proven to be the most successful approach in computer Go and it is rapidly replacing other search algorithms as the method of choice on other domains such as General Game Playing, Amazons, real time strategy games, etc. Computer Go is said to be one of the greatest challenges in AI. The research done on Go playing agents was until recent innovations[1] unable to produce agents that could beat top human players. Despite the fairly simple rules of the territorial board game, the game is extremely hard to master and has been used in the Chinese culture as a measure of intelligence for centuries. The problem when developing efficient agents is a large number of possible moves at each step. Specifically in MCTS, this number coincides with the branching factor, which exponentially increases the search space.

The basic idea of tree search algorithms is to imagine the game as a finite number of possible states, which occur as a result of a move. With MCTS, the states of the game are represented as nodes. New nodes are added to the search tree incrementally and each node contains a value that predicts which player will win the game, according to numerous randomly simulated games from that state. The value of the node can be simply an average outcome of all simulated games.

[1] A British artificial intelligence company named Google DeepMind, founded in 2010, has released a computer program named AlphaGo to play the board game Go. In October 2015 this was the first program to beat a professional human player without handicaps on a $19 * 19$ board.

A search tree is used to guide simulations along promising paths by selecting the child node with the highest value. It is said that the evaluation function continues to improve from knowledge gained by additional simulations and will, given infinite memory and computation time, converge to the optimal solution. In this chapter, we discuss the general MCTS algorithm and put forward the most important variations that evolved through research, mostly done on computer Go programs.

4.1 The general algorithm

The basic MCTS process builds a tree in an incremental asymmetric manner until a predefined computational budget is reached (time, memory) at which point the search is stopped and the best action returned[2]. For each iteration of the algorithm a *tree policy* is used to determine which is the most promising path in the current search tree. This step involves the addition of a leaf node in the search tree. The tree policy is the one balancing exploration (explore areas that have not been sampled yet) vs. exploitation (further exploit areas that appear to be promising). There are multiple strategies for selecting a tree policy, like progressive pruning, simulated annealing, etc. Chaslot et al. published a paper called *Monte Carlo Strategies for Computer Go*, where they are all described in detail [7].

After a path is determined by the tree policy, a simulation is carried out from the leaf node of the path and the values of tree nodes are updated. For simulating games from a certain node, a *default policy* is used. The default policy can be based on random roll-outs, or it can include a more sophisticated strategy of game play. Previous machine learning approaches have focused on optimising the strength of the default policy, under the assumption that a stronger policy will perform better in a Monte Carlo search. According to the survey [5], in practice this assumption is often incorrect, and in general it can be difficult to find a default policy that performs well in the search. What is more, it is important to realize the trade-off between the positive results from including extra logic into the default policy and the resulting increase of resource consumption. To further clarify the idea behind

[2]Note that the child is selected by a specific mechanism: Max child, Robust child, Max-Robust child, Secure child. Detailed description of mechanisms after termination can be found in [5]

the general algorithm, we would like to point out the four steps applied in every search iteration (also shown in Figure 1):

1. Selection: Starting at the root, a child node is selected in accordance with the tree policy.
2. Expansion: One or more child nodes are added to the tree.
3. Simulation: A simulation is run from the new node in accordance with the default policy.
4. Backpropagation: The simulation results are backed up and the values of the nodes are updated.

4.2 Upper Confidence Bound for Trees

The Upper Confidence Bound for Trees (UCT) algorithm treats each state of the search tree as a MAB, in which each action corresponds to an arm of the bandit. The tree policy selects actions by using the UCB1 algorithm, which maximises an upper confidence bound on the values of actions[3]. It is commonly said to be using the so called *optimism in the face of uncertainty* principle with the inclusion of a *bonus* based on an upper confidence bound of the current value. Specifically, the action value is augmented by an exploration bonus that is highest for rarely visited state-action pairs. The tree policy selects the action a^* maximising the value

$$Q'(s,a) = Q(s,a) + c\sqrt{\frac{\ln N(s)}{N(s,a)}} \tag{4.1}$$

$$a* = \operatorname{argmax} Q'(s,a) \tag{4.2}$$

where $Q(s,a)$ is understood to be a scalar within [0,1], $N(s)$ the number of times a current (parent) node has been visited, $N(s,a)$ the number of times action a has taken place and c is a scalar. There is a balance between the first (exploration) and the second (exploitation) term in the equation. When $N(s,a) = 0$, the UCT value $Q'(s,a) = \infty$, so that previously unvisited children are assigned the largest value

[3]We discuss the UCB1 algorithm in Chapter 3 in section Upper Confidence Bound

and are explored immediately. The benefits of MCTS are usually not realised until the algorithm is adapted to suit the domain at hand, hence the performance of UCT can be significantly improved by incorporating domain knowledge in the default policy, as previously stated by Gelly and Silver in [8]. We continue to explore different variations of UCT by describing them briefly and discussing advantages and disadvantages.

4.3 Rapid Action Value Estimation

MCTS separately estimates the value of each state and action, therefore it can not generalize related moves and positions. What is more, in the game of Go the value of a move is often unaffected by the moves placed elsewhere on the board. It is therefore useful to rapidly evaluate a reoccurring move (state-action pair). The Rapid Action Value Estimation (RAVE) uses the AMAF (All Moves As First) heuristics and provides a simple way to share knowledge between related nodes and moves in the search tree. The idea of the AMAF heuristic is to compute AMAF value as a general value for each move, regardless of when it is used. The AMAF value is the mean of all simulation outcomes in which action a is selected at any turn after state s is encountered, as noted by Gelly and Silver in [8]. In RAVE instead of computing the UCT value, the state-action pairs are evaluated following the equation 4.3

$$\bar{Q^{\gamma}}(s,a) = Q^{\gamma}(s,a) + B(\bar{s},a), \tag{4.3}$$

where $Q^{\gamma}(s,a)$ is the AMAF value and $B(\bar{s},a)$ is the level of bias, which depends on the individual pair. Note that assuming the values of moves are truly independent, this rough value estimation provides a much faster extension of the algorithm and therefore the agent gains more information. Unfortunately, the algorithm can often be wrong when evaluating the moves, precisely because of the independence of moves assumption. More often, a move will not have the same effect on the game state at different stages of a game, hence the same evaluation of such moves is false. A good example would be to consider the SoC - a simple strategy will favour building roads and settlements in the beginning of the game in order to gain power over the territory and resources. The Monte Carlo RAVE algorithm

settles this issue by combining the calculation of UCT and AMAF value with a weighted sum as follows

$$Q_*(s,a) = (1-\beta)(s,a)Q(s,a) + \beta(s,a)Q(\bar{s},a), \tag{4.4}$$

where $\beta(s,a)$ represents a weight for a state-action pair, $Q(s,a)$ represents the MC value[4] and $Q(\bar{s},a)$ the AMAF value.

By incorporating the *optimism in the face of uncertainty* principle, the so called UCT-RAVE algorithm is yet another extension that combines the AMAF and UCT values for move evaluation. The algorithm, similarly to MC-RAVE, takes advantage of the rapid estimates, while also using the UCT evaluation of moves following the equation 4.1. In contrast, the algorithm uses a schedule to determine the evaluation method, rather that a weighted sum. Intuitively, when the schedule decreases to zero, the algorithm becomes equivalent to UCT. According to Gelly and Silver [8], typically a hand-selected schedule is applied, yet there has also been an attempt to derive a statistical model for MC-RAVE. Another development was the usage of a Minimum Squared Error (MSE) schedule, which assumes that both AMAF and MC values are Bernoulli random variables.

Rapid action value estimation and its variants have been researched in the context of the game of Go, where the state-action pairs are not necessarily unique throughout the search tree. On the other hand, the search tree built in the domain of the SoC contains only unique state-action pairs, since the pieces, once placed on the board, can not be removed until the end of the game. Therefore, only when all players subsequently pass the move and the next player passes the move again, the state-action pair could be regarded as not unique.

4.4 Heuristic prior knowledge

In order to further improve the performance and reduce uncertainty for rarely encountered positions, Gelly and Silver describe how they incorporate prior heuristic knowledge by using a heuristic function $H(s,a)$ and a heuristic confidence function $C(s,a)$. When a new node is added to the tree, it is initialized according to the

[4]MC value is the mean outcome of all simulations in which action a was selected at state s.

heuristic function $Q(s, a) = H(s, a)$ and $N(s, a) = C(s, a)$. After the initialization the values are updated normally as they were in Monte Carlo simulation. Heuristic was applied to MC-RAVE and UCT-RAVE, but according to their measurements, heuristic MC-RAVE outperformed heuristic UCT-RAVE.

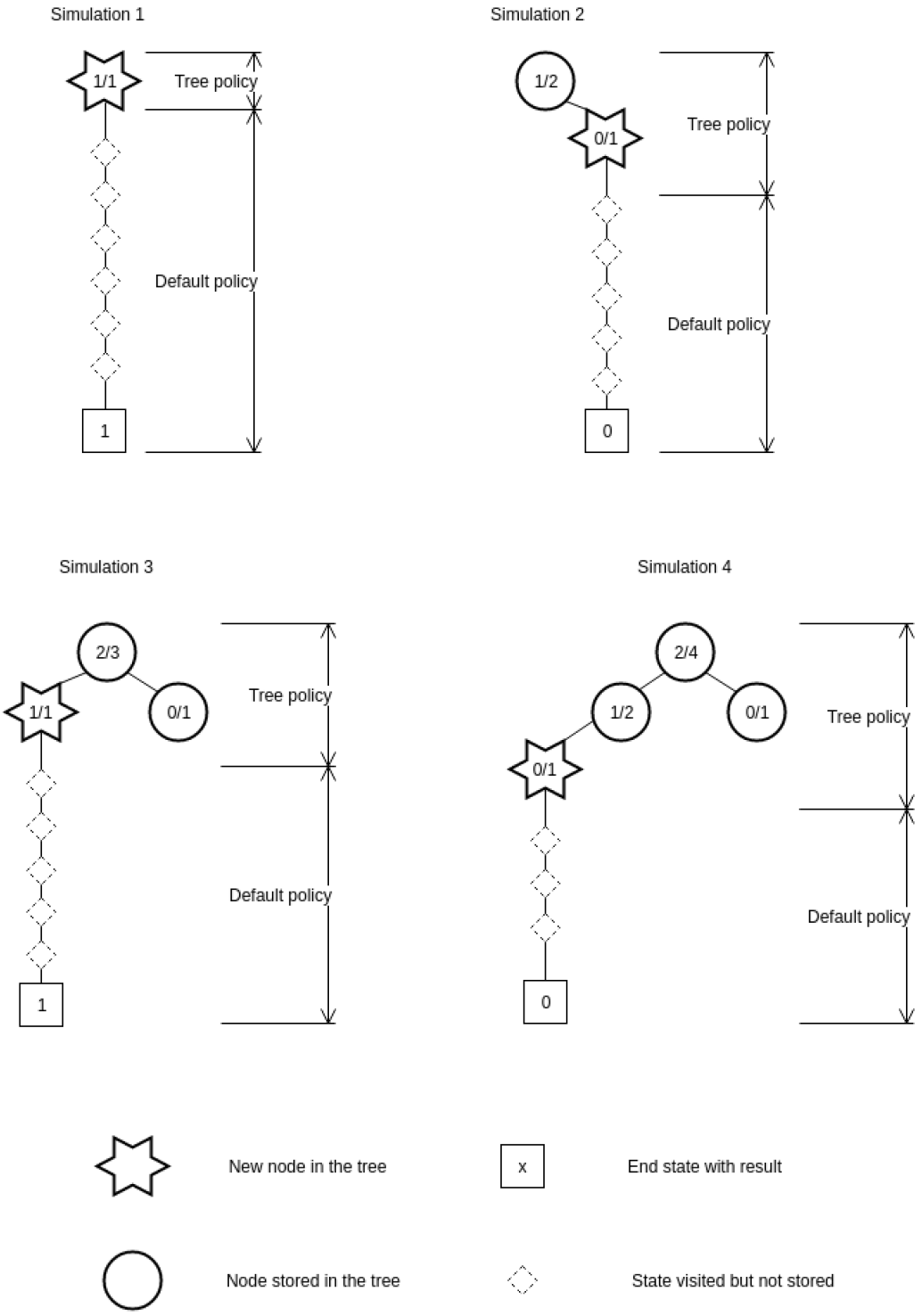

Figure 1: Simulations of a Monte Carlo tree search.

Chapter 5

Thompson sampling in Monte Carlo Tree Search

Thompson sampling (TS) has become in the recent years a very popular method of web advertisement and has been continuously applied in many different areas to solve complex decision problems and planning. Empirical studies show that TS is able to achieve similar or sometimes even better performance than other types of algorithms in practice. Nevertheless, the research community appeared to be reluctant in applying TS, due to the lack of theoretical analysis and proof of convergence. The purpose of this chapter is to discuss selected related work concerning TS and its applications in MCTS, reflect upon its application in the SoC and discuss our program and its contributions in detail.

5.1 Preliminary investigations

Agrawal and Goyal [1] have shown that the TS algorithm achieves logarithmic expected regret for the stochastic MAB problem. Research also shows that there have been further developments in theoretical analysis of the algorithm, in fact Kaufmann et al. published the first proof of the asymptotic optimality of Thompson sampling for Bernoulli bandits in 2012 [10]. Furthermore, in research paper [3], Bai et al. presented a novel approach for MCTS using Bayesian mixture modelling and inference based Thompson sampling and apply it to the problem of online planning

Table 5.1: The hidden probabilities of arms' rewards in a test MAB environment, where ϵ varies.

Arm 0	Arm 1	Arm 2	Arm 3	...	Arm 8	Arm 9
0.5	0.5-ϵ	0.5-ϵ	0.5-ϵ	...	0.5-ϵ	0.5-ϵ

in MDPs. Their experimental results show that their approach beats the state-of-the-art UCT approach. What is more, very good results have been obtained by Wu et al. in their work proposing the Double Thompson sampling for Dueling Bandits [20]. The aforementioned authors present the regret analysis and provide regret measurements demonstrating the efficiency of the proposed solution.

Following the results of An Empirical Evaluation of Thompson sampling published by Chapelle and Li [6] we have similarly tested the performance of TS in comparison to UCB on a trivial MAB problem. Similarly to the authors, we have obtained very positive results, demonstrating that the regret of TS does indeed increase slower than the regret[1] of UCB. The measurements were done using a simple model of a 10-armed bandit. The hidden winning probabilities of individual arms are presented in Table 5.1. From Figure 2 we deduce that TS only requires about 1000 simulations to outperform the UCB. Note that the regret is averaged over 10 trials.

As TS has, to the best of our knowledge, never been implemented in the tree policy of MCTS nor tested on the board game of SoC, we sought to explore the possibility of our program to outperform the standard UCT algorithm. Instead of evaluating moves according to the UCB value, our program selects random samples of each arm from a posterior Beta distribution, which is updated accordingly. The Beta distribution was selected because it is a conjugate prior distribution to the Bernoulli distribution. It is a continuous probability distribution, defined on the interval [0,1] and parametrized by positive real shape parameters α and β. We incorporate TS in the tree policy of MCTS as is presented in Algorithm 2, where S_i represents the number of successful plays of arm i, F_i the number of failures

[1]Note that the cumulative regret is calculated as a difference between the winning probability of a chosen arm and the winning probability of the optimal arm at time T.

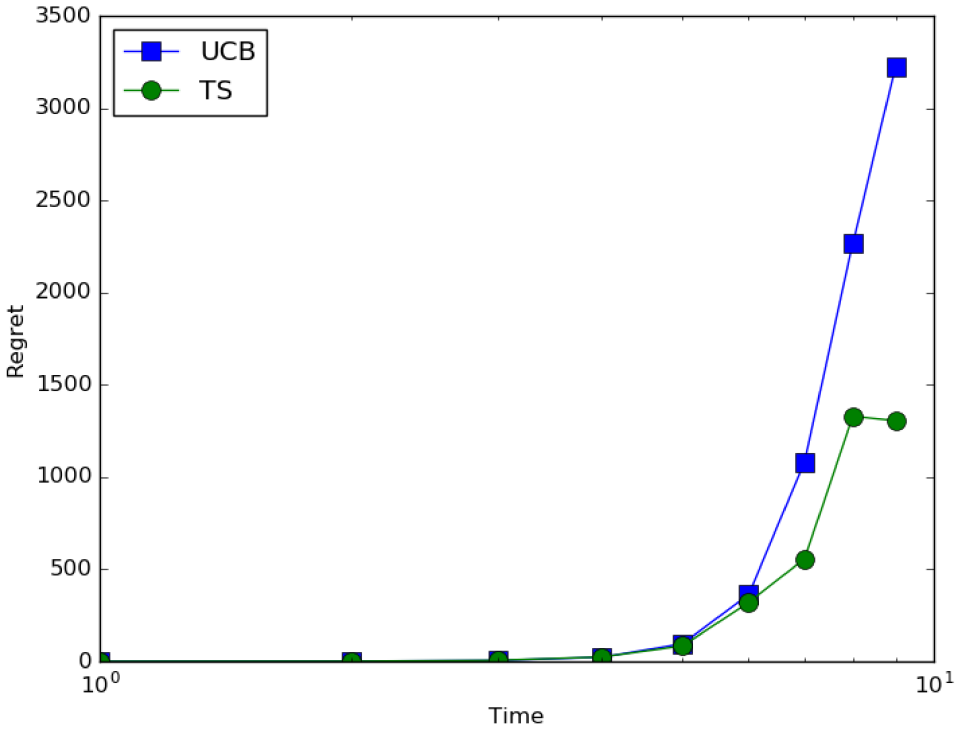

Figure 2: Regret of choosing non-optimal arms measured on a trivial MAB problem, where number of arms is 10 and $\epsilon = 0.1$. Measurements have been taken for $T = 4^0, 4^1, 4^2, ...4^8$ arm pulls, averaged over 10 trials.

and α and β are set to 1 to start with the Uniform distribution.

However promising TS might seem in MAB problems, we would like to point out that the domain of our program is different from the ones considered in this section. We continue to describe the structure of our program and performance measurements.

5.2 Structure of our program

Our main program is entirely implemented in the programming language C. We separately implemented the MCTS-UCT and the MCTS-TS algorithms on simple games, namely Sum Of Switches (SOS) and Gomoku with a varied board size, for

Algorithm 2 MCTS with Thompson sampling in the Tree Policy

$\alpha \leftarrow 1, \beta \leftarrow 1, S_i \leftarrow 0, F_i \leftarrow 0$
procedure TS SEARCH
  Create a root node v_0 with game state s_0
  while within computational budget **do**
    $v_1 \leftarrow$ Tree Policy(v_0)
    $\Delta \leftarrow$ Default Policy($s(v_1)$)
    Backup(v_1, Δ)
  end while
return a(Best Child(v_0))
end procedure

procedure TREE POLICY(v)
  while v is non-terminal **do**
    if v not fully expanded **then return** Expand(v)
    else
      $v \leftarrow$ Best Child(v)
    end if
  end while
end procedure

procedure BEST CHILD(v)
  for every child i of v **do**
    Draw θ_i according to Beta($S_i + \alpha, F_i + \beta$)
  end for
  Select the child with max θ_i
end procedure

```
procedure DEFAULT POLICY(s)
    while s is non-terminal do
        randomly choose valid action a
        s ← f(s, a)
    end while
return reward for state s
end procedure

procedure BACKUP(v, Δ)
    v_i ← v
    while v_i is not null do
        if Δ is a win then
            S_i + +
        else
            F_i + +
        end if
        v_i ← parent of v_i
    end while
end procedure
```

the purpose of testing the performance of MCTS on more trivial domain examples. Initial measurements on MCTS-based implementations in trivial game situations have shown the trend of increasing performance compared to random moves. After the implementation of the SoC and its main components[2], we were able to incorporate the game logic into the MCTS-based implementation. What is more, we have implemented two basic strategy playing agents, in order to have a more realistic comparison of the agent's strengths.

5.3 Basic strategy playing agents

We have chosen to implement two agents that are engaging to play the game using a particular strategy. To further illustrate, we continue to describe the two most commonly adopted strategies of game play.

5.3.1 Expand early: clay & wood

In compliance with the rules noted in Chapter 2, a player might seek to expand his/her territory early in the game so as to gain more control over various resources. This strategy aims to start buying Development Cards, obtaining the longest road or largest army later in the game. However, in order to build settlements it is absolutely necessary to build roads first, hence the player should begin the game by positioning himself next to clay and wood resources, while keeping in mind the desired variety of resources for immediate expansion. Finally, building a city requires prior existence of a settlement, therefore the player seeks to build cities later in the game as well, depending on the available resources.

5.3.2 Develop more than grow: ore & wheat

Another possible and effective strategy is for the player to develop fairly soon in the game and try building as many cities as possible. Furthermore, this strategy favours playing Development Cards, which eventually leads to largest army. In the SoC it is understood that there is a shortage of ore and clay on the board,

[2]We have also included a basic GUI to enable testing the program's strength against a human player.

which increases the importance of initially positioned settlements to border ore producing tiles, when playing this strategy.

Our agents were developed to behave similarly to the above described game strategies. We would like to stress that in practice, the initial set-up has an enormous effect on the performance of agents, as previously studied by Gelly and Silver, hence our agents (MCTS, as well as strategy agents) are playing intelligently from the start of the game. To further clarify, this means, that MCTS agents perform the predefined number of simulations in the first phase of the game as well. The strategy playing agents, on the other hand, play according to a fixed strategy for placing initial pieces. Note that strategy playing agents have been developed to also favour positioning initial settlements on vertices bordering good resource production tokens, i.e. 6 and 8.

5.4 Testing against human players

In this section we briefly describe the design of a minimized user interface, built to enable testing the program against a human player, and the observations obtained from testing the agents. Figure 3 shows the representation of the game board after the console prompt for user input. Twenty games have been played against the developed agents allowing 4000 simulations per a MCTS move, with a 17/20 win rate of the author. In most cases, MCTS-based agents obtained more points than strategy playing agents. In fact, we have observed that the MCTS-UCT agent has mostly performed second best. Note that, the performance was estimated not only in the number of points obtained, but also in the amount of built pieces on the board and bought Development Cards, which gives a player advantage in the following rounds. Our observations of the performance of agents are concurrent with the obtained regret and average win rate measurements presented in Section 5.5. As expected, an experienced human player is still superior to the agents. Not only does an experienced human have better abilities to strategically plan the moves, the player has also an overview of the current game state, including resource and Development Cards of opponents.

For the purpose of faster performance measurements, we have altered MCTS-based agents to only have 1 out of 7 possible moves at a given time:

1. trade,
2. build road,
3. build settlement,
4. build city,
5. buy a Development Card,
6. play a Development Card,
7. and pass the turn.

This alteration changes the game, as in the SoC it is expected to make multiple moves when possible. We have tested the agents again, this time allowing the human player to make only 1 move per turn, removing the advantage of multiple actions from human agents, so that all agents behave similarly. We have observed similar results, reaching a 16/20 win rate for the human player. Nevertheless, the number of rounds per game has noticeably exploded, an average game taking about 100 rounds to finish. Note that even for a human player, it became more difficult to plan moves, as the game logic forced the player to only make one move per turn. Various game plays that are in practice very useful, have become impossible to perform.

To further illustrate, such a scenario may occur when a player is preparing to play the Development Card of type Year of plenty. This card is often played when the player lacks one resource in particular and therefore seeks to play this card only when he has other sufficient resources to do several moves. Imagine if you will, that a player possesses the following Resource Cards: 1 clay, 1 wood, 1 sheep, 1 wheat and 3 ore. Furthermore, imagine that the player had obtained 8 Victory Points and the Development Card of type Year of Plenty in previous rounds. If the player were able to play the game according to the official game rules, assuming the game state allows this player to build another settlement, he/she would win performing the following:

1. Play the Development Card of type Year of plenty and gather 2 Resource Cards of type wheat from the supply stack,

2. Build another settlement and obtain 1 extra Victory Point,

3. Build a city on one of the previous settlements and obtain another Victory Point, consequently winning the game with 10 points.

Being forced to only make 1 move per turn, introduces a level of uncertainty that the player will keep his resources until the next turn. One might argue that this game play can be divided into three turns. In the first turn, the player builds the settlement, which also reduces the risk of loosing resources. In the second turn the player plays the Development Card and in the third turn the player builds the city - that is if the resources were not lost during the last turn. Not only is there increased risk of loosing the resources needed in each following turn, there is also a greater chance of opponents winning the game in the meantime. Indeed, this kind of game play was adopted while testing, yet there is a significant disturbance caused by the limited game rules, which consequently lead to the deduction, that such limitations are too large and change the game significantly.

With further alterations to the primarily developed TS agent, we were able to increase its performance compared to the state-of-the-art UCT approach. While testing the agents against a human player, we were able to observe similar winning rates for the human player, however, the strength of the TS agent increased accordingly. We further discuss the performance of the altered TS agent in the following section.

5.5 Performance measurements

Finally, we present the performance measurements of our implementation of TS in MCTS against other agents. While performing runs of multiple games with a varied number of possible MCTS simulations per move, we have been taking note of the individual agent's average win rate and regret[3]. The measurements were performed primarily on agents with the possibility of making multiple moves in one turn, secondly on agents allowed to make a single move at each turn and thirdly

[3]Notice that in comparison to the regret measured in the empirical evaluation in Section 5.1, this regret is calculated as a relation between the number of lost and won games at time t. The regret is normalized according to the number of games measured.

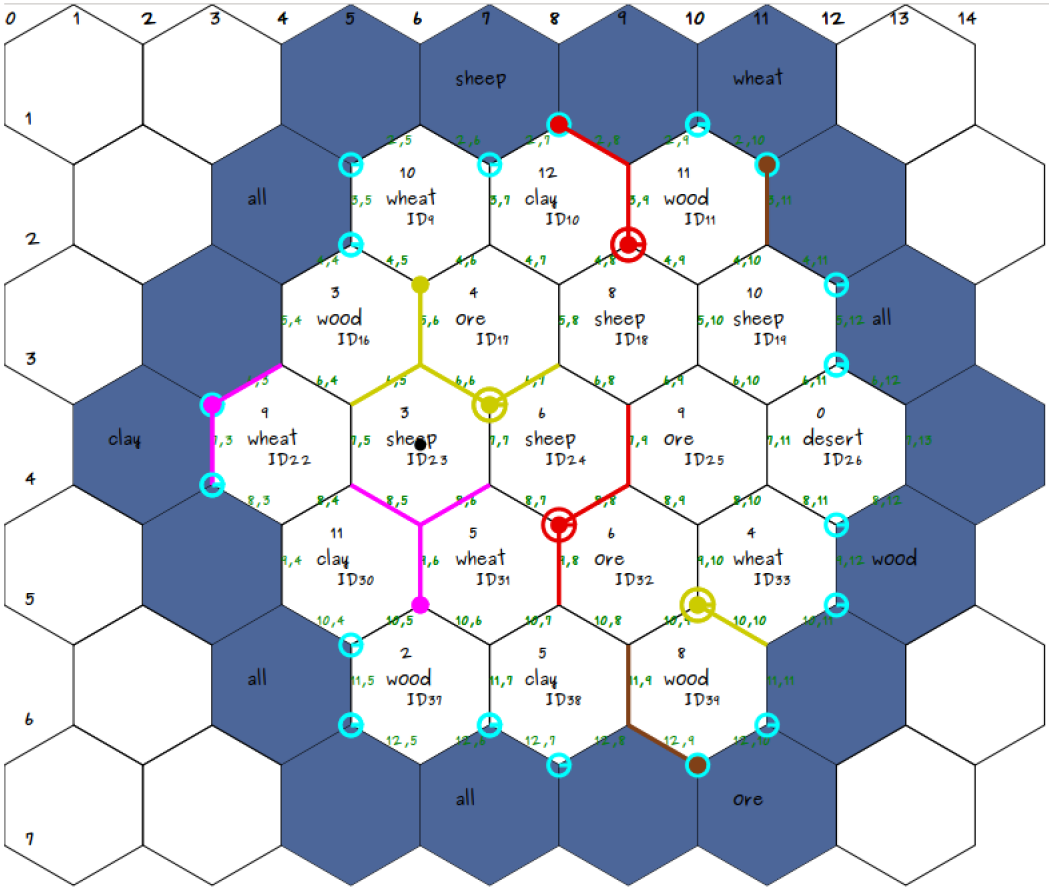

Figure 3: Example of a game state.

on initially developed agents with the altered TS agent. The aforementioned measurements are presented in figures 7, 8, 9, 10, 11, 12 and are further discussed in the following subsection.

5.5.1 Discussion of results

Initial results of regret and average win rate measurements, presented in figures 9 and 10, reflect the playing strength of individual agents. There is a noticeable proportional growth of the MCTS-based agent's performance with the number of simulations allowed per a MCTS move, as expected. What is more, it is observed that the performance of strategy playing agents decreases significantly when playing against the MCTS-based agents. In fact, if enough simulations per a MTSC

move are allowed, the strategy playing agents never win. On the other hand, the results show a consistent defeat of the MCTS-TS agent by the MCTS-UCT agent.

As previously mentioned, an alternative version of the program has been developed, where the focus was set on decreasing the branching factor, e.i. possible moves per turn, to enable learning on trees developed in depth, rather than learning from trees with a maximum depth of 3, at best. Initially, our reasoning led us to develop a single move as a combination of 6 basic moves: trade, build a road, build a settlement, build a city, buy a Development Card, and play a Development Card. Since multiple moves are allowed, this results in $2^6 = 64$ possible moves at each turn, evidently causing a branching factor of 64. The alternative version of the program, therefore includes modified agents where only one of the basic moves is allowed, decreasing the branching factor efficiently to 7. Unfortunately, this caused a decrease in the overall performance of all agents, which we understand to be present, due to the significant modification of removing the possibility of combined moves. In the SoC, the ability to trade and make another move in the same turn is precisely how the game advances. As our initial implementation includes the possibility of combined moves per turn, the possible moves were not calculated in advance. After the tree policy, a random combination of basic moves was decided upon before it was confirmed to be possible. This resulted in a number of non-beneficial simulations, which did not greatly damage the overall performance of the program. With the alternative version, however, we had to include the prior calculation of possible moves, without which the game developed very slowly. Typically, the game ended in about 30 to 60 rounds, without the additional calculations it ended only after 150 to 300 or more. Finally, the score required for the end of the game was reduced to 7. The alternative version of the program manages to compute the same amount of games and number of tree simulations in half of the time, yet MCTS-UCT agent remains superior to MCTS-TS agent. Final regret and average win rate measurements are presented in figures 11 and 12. As previously discussed, we have reason to believe that such alterations of game rules change the game significantly and we therefore continue to address the initial version of the program.

Previously demonstrated by the aforementioned empirical studies in Section 5.1, TS is proven to be efficient in the domain of a classical MAB problem, therefore

the decreased performance might exist due to the domain differences. Furthermore, we point out that in comparison with the classical MAB problem, the hidden winning probabilities of nodes might vary throughout the search space, whereas in the supporting study they remain unchanged. To further clarify, consider the fact that building roads towards the end of the game, while already fully expanded, might be regarded as a loss of resources, while doing so in the beginning brings greater success. Having said that, we have made further alterations to the TS node evaluation to include the information gained with multiple MCTS simulations. As previously studied by Chapelle and Li in [6], the attempt to increase performance of the MCTS-TS agent was wade by introducing *posterior reshaping* and drawing samples from a modified distribution. We have modified the evaluation of nodes to sample the Beta distribution as presented in Algorithm 3, where ϕ represents the aforementioned exploration bonus $\sqrt{\frac{\log N(s)}{N(s,a)}}$ in Section 4.2.

Algorithm 3 Modified Thompson sampling node evaluation

procedure BEST CHILD(v)
  for every child i of v **do**
    Draw θ_i according to Beta($\frac{S_i+\alpha}{\phi}$, $\frac{F_i+\beta}{\phi}$)
  end for
  Select the child with max θ_i
end procedure

By including the exploration bonus in the positive shape parameters of the Beta distribution, the drawn samples devalue the nodes causing the increased behaviour of exploring instead of exploiting the tree. To further demonstrate the behaviour of posterior reshaping, we present the figures of a Beta distribution when $\phi = 0.5, 1.0, 1.5$ in figures 4, 5 and 6. We present the regret and average win rate measurements of individual agents when using the altered TS agent in Figures 7 and 8. Regret measurements reflect a similar performance of UCT compared to TS. With increasing the number of possible simulations, however, the TS agent outperforms the UCT agent.

In the future, our work may be extended in many ways including the full implementation of game logic with Domestic Trade and better strategy playing agents

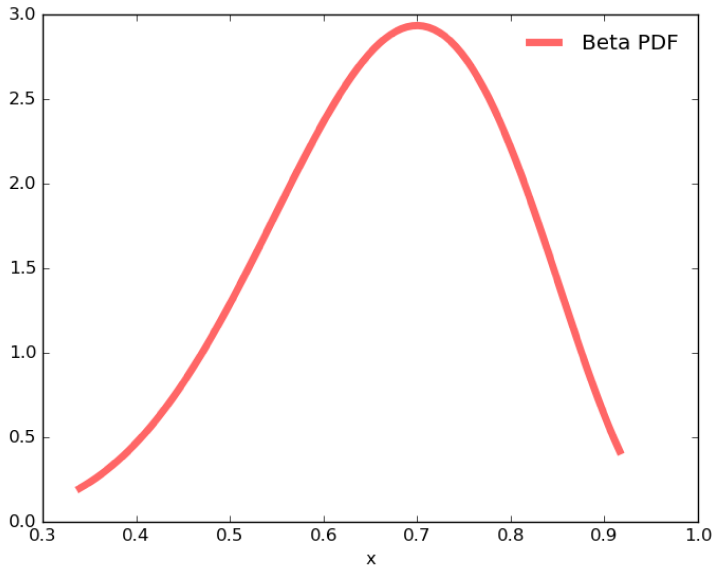

Figure 4: The PDF of a reshaped Beta distribution when $\phi = 0.5$, $S_i = 40$ and $F_i = 1$.

with stronger performance results. Furthermore, we propose an implementation of an interface module, in order to test the performance of agents against the open source implementation of Java based playing agents, namely JSettlers.

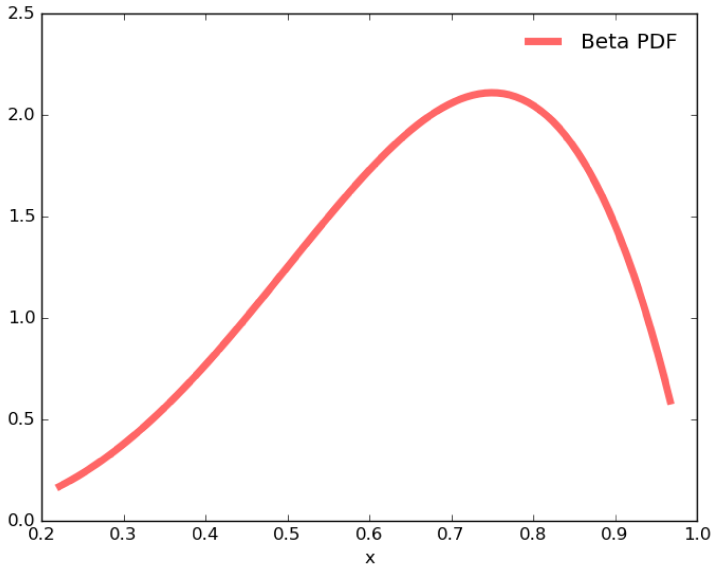

Figure 5: The PDF of a reshaped Beta distribution when $\phi = 1.0$, $S_i = 40$ and $F_i = 1$.

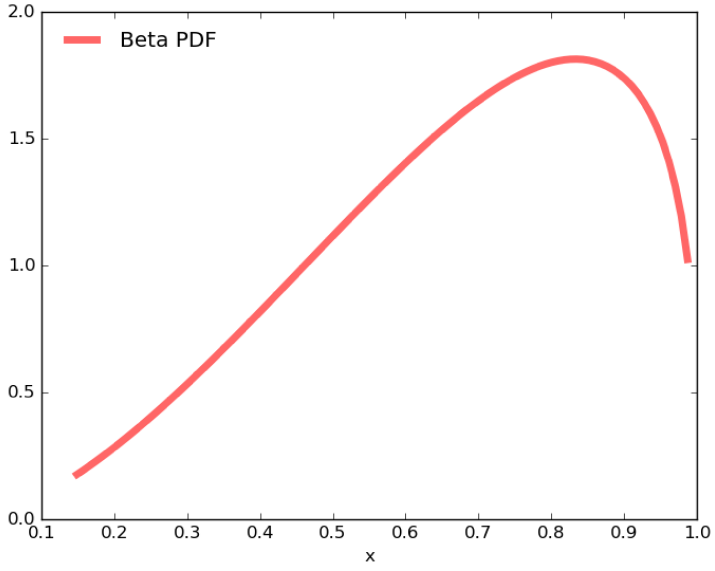

Figure 6: The PDF of a reshaped Beta distribution when $\phi = 1.5$, $S_i = 40$ and $F_i = 1$.

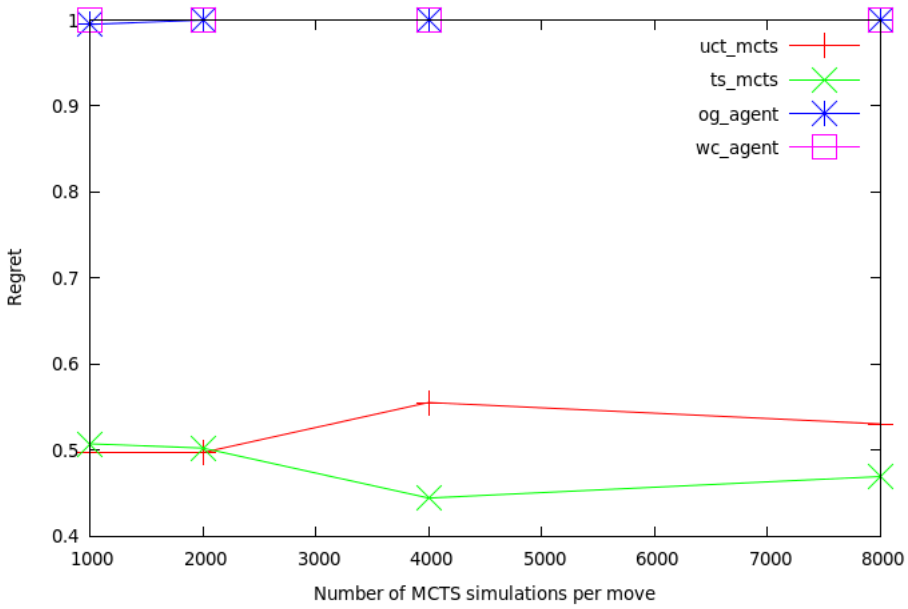

Figure 7: Regret of agents playing against the altered MCTS-TS agent including combined moves per turn and posterior reshaping.

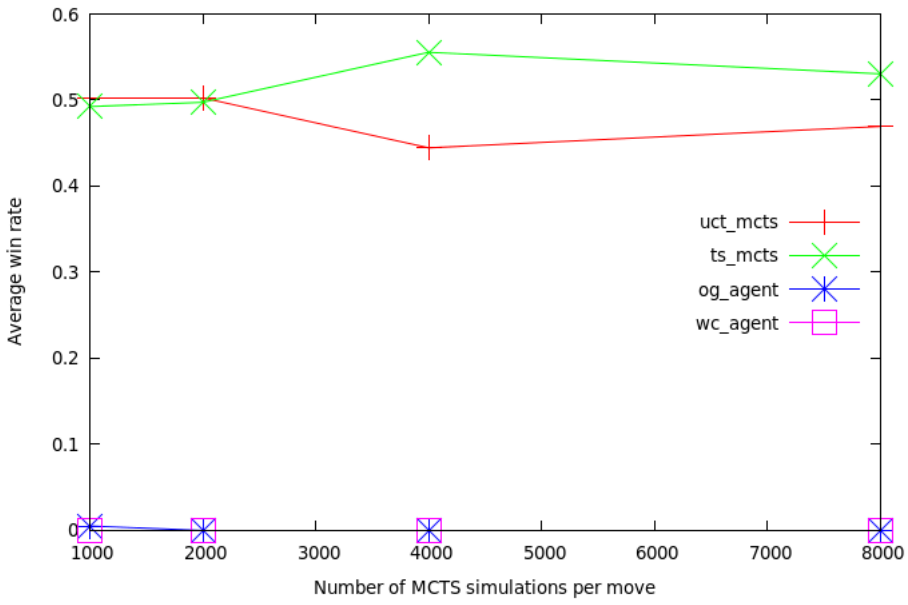

Figure 8: Average win rate of agents playing against the altered MCTS-TS agent including combined moves per turn and posterior reshaping.

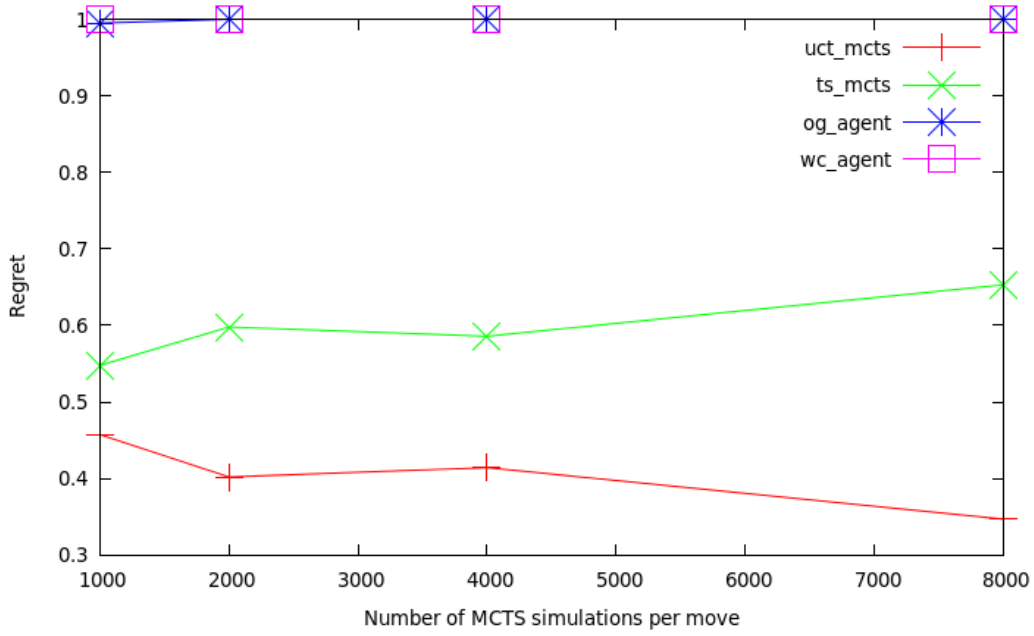

Figure 9: Regret of agents playing against the primarily developed MCTS-TS agent including combined moves per turn.

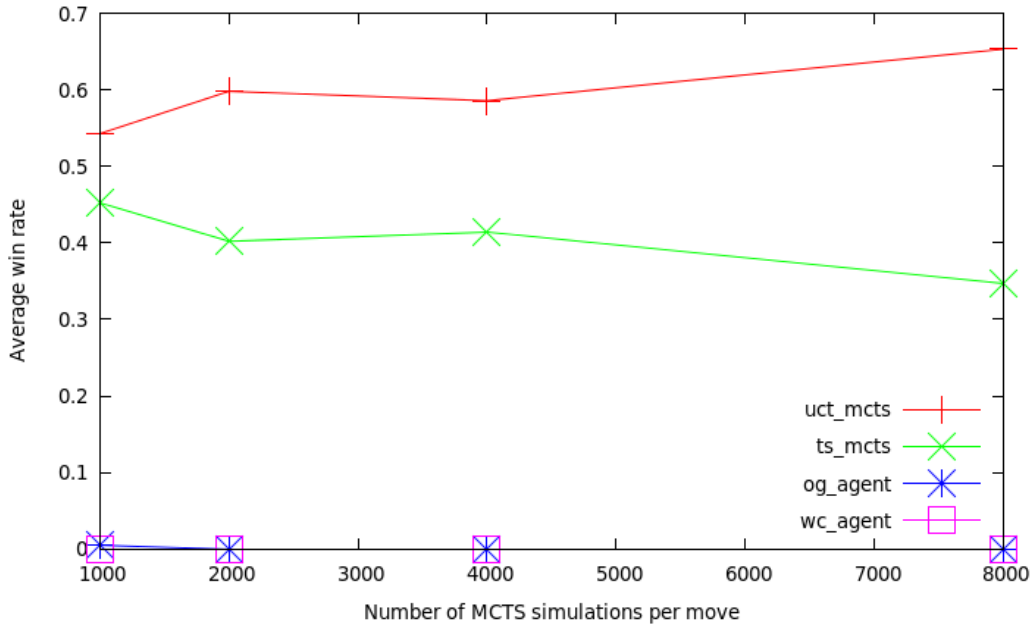

Figure 10: Average win rate of agents playing against the primarily developed MCTS-TS agent including combined moves per turn.

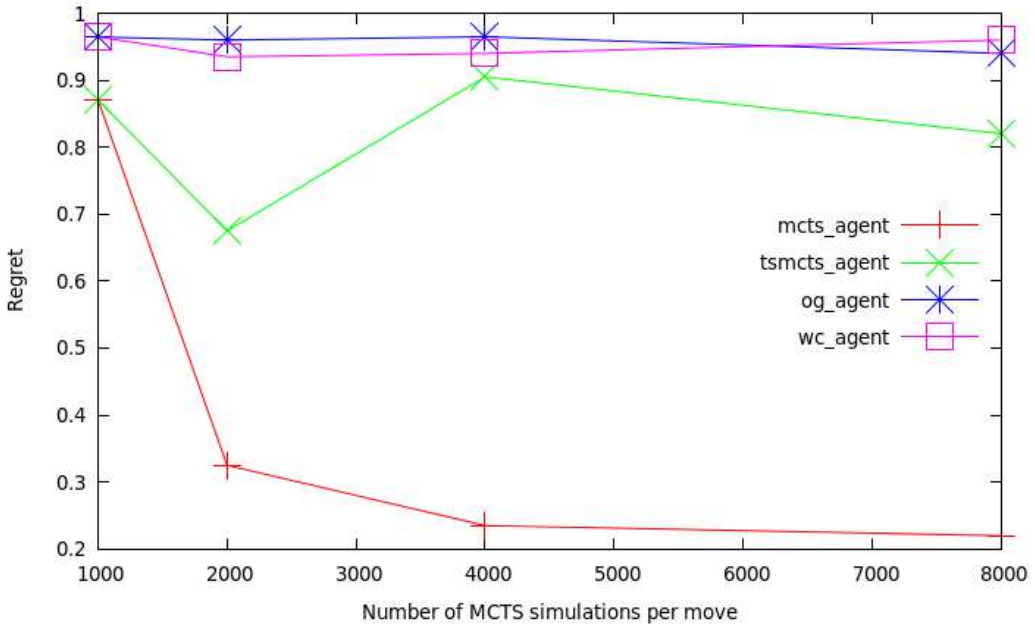

Figure 11: Regret of agents playing against a MCTS-TS agent without combined moves per turn or posterior reshaping.

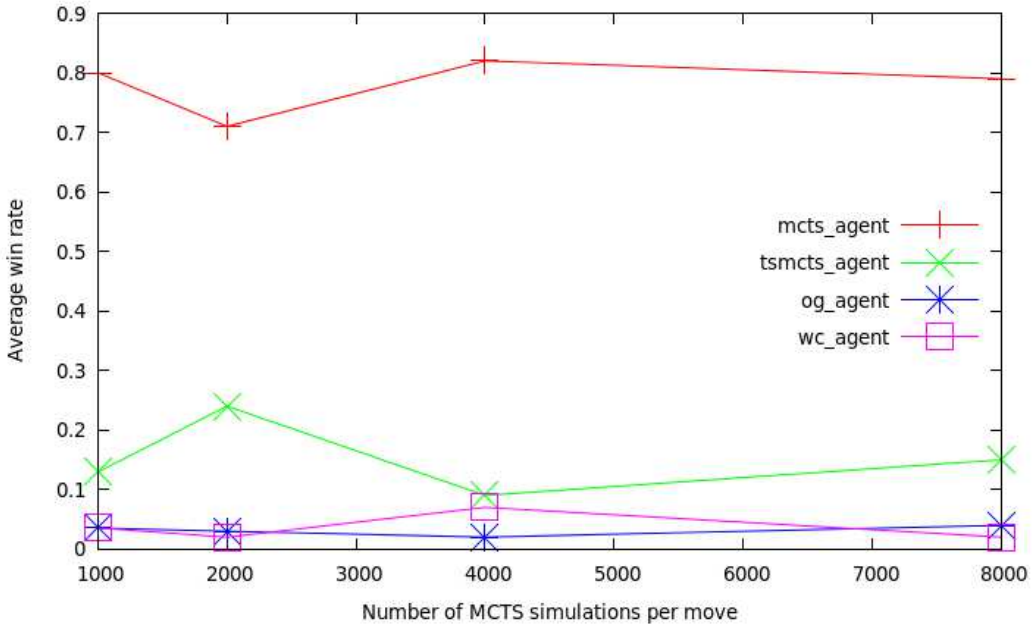

Figure 12: Average win rate of agents playing against a MCTS-TS agent without combined moves per turn or posterior reshaping.

Chapter 6

Conclusion

This Master's Thesis introduces a novel approach into the tree policy of the MCTS algorithm. We have successfully implemented a TS-based approach not only in trivial games such as Gomoku, but also in a new domain, namely the board game the Settlers of Catan. Our program is implemented in C, with certain limitations regarding the rules of the game; namely, we have excluded the trading between agents and limited the game play to four agents at a time. We have made some further modifications for the purpose of acquiring measurements faster, which have had some negative effects on the MCTS-based agents' performance. We have made further alterations to the TS agent, including the posterior reshaping of distribution, which increased the performance of TS agent significantly, causing it to outperform the state-of-the-art UCT approach as the number of allowed simulations per move increases.

Our research also includes the study of different algorithms approaching a MAB problem. After the study of the Gittins index, UCB, TS, UCT, RAVE and its varieties, we deduced that modifying the tree policy of MCTS may result in promising results. What is more, we have found supporting studies documenting the empirically measured superiority of TS over UCB on a classical MAB problem. Hence, we chose to implement one of the most popular algorithms applied in online advertising, TS, into the tree policy of MCTS and measure its performance against the state-of-the-art UCT algorithm on the board game the Settlers of Catan. Since, to the best of our knowledge, such an approach has never been studied before, we present this as our main contribution to the field.

Bibliography

[1] S. Agrawal and N. Goyal. Analysis of Thompson sampling for the Multi-armed Bandit problem. *arXiv preprint arXiv:1111.1797*, 2011.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the Multi-armed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[3] A. Bai, F. Wu, and X. Chen. Bayesian mixture modelling and inference based Thompson sampling in Monte-Carlo tree search. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2013.

[4] L. Branca and S. J. Johansson. Using multi-agent system technologies in Settlers of Catan bots. *City*, 2:3, 2007.

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[6] O. Chapelle and L. Li. An empirical evaluation of Thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257, 2011.

[7] G. Chaslot, J.-T. Saito, B. Bouzy, J. Uiterwijk, and H. J. Van Den Herik. Monte-Carlo strategies for computer Go. In *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91. Citeseer, 2006.

[8] S. Gelly and D. Silver. Monte Carlo tree search and Rapid Action Value Estimation in computer Go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[9] D. M. Jones and J. C. Gittins. *A dynamic allocation index for the sequential design of experiments*. University of Cambridge, Department of Engineering, 1972.

[10] E. Kaufmann, N. Korda, and R. Munos. Thompson sampling: An asymptotically optimal finite-time analysis. In *International Conference on Algorithmic Learning Theory*, pages 199–213. Springer, 2012.

[11] V. Kuleshov and D. Precup. Algorithms for Multi-armed Bandit problems. *arXiv preprint arXiv:1402.6028*, 2014.

[12] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[13] A. Mahajan and D. Teneketzis. Multi-armed Bandit problems. In *Foundations and Applications of Sensor Management*, pages 121–151. Springer, 2008.

[14] J. C. Mellor. Decision making using Thompson sampling. 2014.

[15] M. Pfeiffer. Reinforcement learning of strategies for Settlers of Catan. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, 2004.

[16] G. Roelofs. Monte Carlo tree search in a modern board game framework. *research paper available at umimaas. nl*, 2012.

[17] I. Szita, G. Chaslot, and P. Spronck. Monte-Carlo tree search in Settlers of Catan. In *Advances in Computer Games*, pages 21–32. Springer, 2009.

[18] J. N. Tsitsiklis. A short proof of the Gittins index theorem. *The Annals of Applied Probability*, pages 194–199, 1994.

[19] R. Weber et al. On the Gittins index for Multiarmed bandits. *The Annals of Applied Probability*, 2(4):1024–1033, 1992.

[20] H. Wu, X. Liu, and R. Srikant. Double Thompson sampling for Dueling Bandits. *arXiv preprint arXiv:1604.07101*, 2016.