Monte Carlo Tree Search for the Game of Diplomacy

Alexios Theodoridis
atheodoridis@isc.tuc.gr
School of Electrical and Computer Engineering,
Technical University of Crete
Chania, Greece

Georgios Chalkiadakis
gehalk@intelligence.tuc.gr
School of Electrical and Computer Engineering,
Technical University of Crete
Chania, Greece

ABSTRACT

Monte Carlo Tree Search (MCTS) is a decision-making technique that has received considerable interest in the past decade due to its success in a number of domains. In this paper, we explore its application in the "Diplomacy" multi-agent strategic board game, by putting forward and evaluating eight (8) variants of MCTS Diplomacy agents. In the core of our MCTS agents lies the well-known *Upper Confidence Bounds for Trees (UCT)* bandit method, which attempts to strike a balance between exploration and exploitation during the search tree creation. Moreover, we devised a heuristic weighting system for prioritizing the tree nodes' actions, and used it to effectively incorporate high-quality domain knowledge in some of our agents. We provide a thorough experimental evaluation of our approach, in which we systematically compare the performance of our agents against each other and against other opponents, including the state-of-the-art Diplomacy agent, *DBrane*. Our results verify that several of our agents are highly competitive in this domain, exhibiting as they do performance which is comparable to, and in some instances superior to, that of *DBrane*. Interestingly, the MCTS approach consistently outperforms all others in tournaments in which one MCTS agent faces one *D-Brane* agent and several other opponents.

KEYWORDS

Monte Carlo Tree Search (MCTS), Upper Confidence Bounds for Trees (UCT), Multi-agent Strategic Board Games

1 INTRODUCTION

Monte Carlo Tree Search (MCTS) is a collective name for a family of methods seeking to identify optimal decisions in a given domain, while making use of the results of simulated outcomes in the search space [1]. It has received considerable attention since its noteworthy success at the game of computer Go [11], and at the same time it has been effective in a variety of other domains [1]. What makes MCTS really effective is its ability to combine the accuracy of a tree search with the generality of random playouts. It is a promising AI game-playing approach, especially for domains that can be represented as trees of sequential states.

SETN 2020, September 2–4, 2020, Athens, Greece

ACM ISBN 978-1-4503-8878-8/20/09…$15.00
https://doi.org/10.1145/3411408.3411413

MCTS assumes that the actual actions' values may be approximated using random simulation; and then employs these values in an efficient manner to adjust the policy towards a best-first strategy [1]. The family of MCTS algorithms consists of a significant number of variations of tree policies. Each of them tries to handle in an efficient way the dilemma between the exploitation of the currently gained knowledge about the actions and the exploration of the search space in order to improve that knowledge. In our work here we use the perhaps most popular tree policy, the *Upper Confidence Bound for Trees (UCT)* one [13].

In this paper we employ MCTS in the Diplomacy [6] domain. Diplomacy is a multi-agent strategic game played by seven players, and requires quite developed strategic and negotiation skills. It is a game of high complexity; though there is full state and actions observability, the players move simultaneously. This renders any attempted MCTS implementation challenging, because each action (i.e., a player's "move order" for this domain) has to be combined with the unknown simultaneous actions of six other opponents beforehand, thus resulting in a non-predictable outcome. Moreover, representing this domain as a tree is challenging on its own right, because of the extremely high number of joint actions that can be taken at each state of the game. There is a need for balancing any pruning of the game tree (which is required in order to reduce the search space to a manageable size), and the benefits of the generality of random sampling that MCTS offers. In our work, we attempt to strike this balance, with our experimental results indicating we did so successfully.

As mentioned earlier, at the core of our MCTS approach lies the well-known *Upper Confidence Bounds for Trees (UCT)* bandit method, which attempts to balance between exploration and exploitation at the creation of a search tree. We enhance this basic MCTS framework in several ways, putting forward eight (8) different versions of the algorithm. These MCTS variants essentially differ in the amount and quality of domain knowledge incorporated in each one of them. We attempted to keep this domain knowledge at a minimum, while at the same time making the approach worthwhile in terms of time required for effective decision-making. Regardless, we devised a heuristic system for sorting the nodes' actions, and equip some of our agents with it. Our experiments verify that injecting an MCTS agent with such high-quality domain knowledge, can significantly boost its performance.

To the best of our knowledge, this is only the second time in the literature that Monte Carlo Tree Search has been employed in the game of Diplomacy, the first one involving some simple experimentations with a very basic MCTS method in the context of a Master's thesis [4]. Importantly, we provide a framework upon which further research on employing MCTS in the game of Diplomacy can

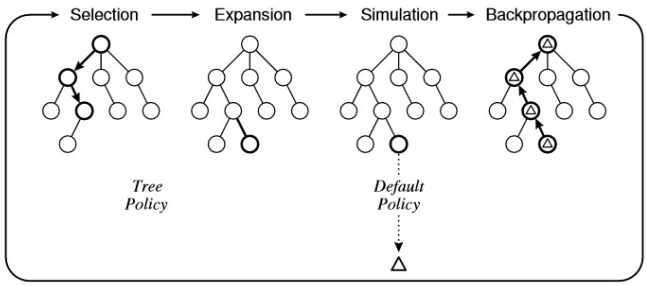

Figure 1: One iteration of the generic MCTS approach [1]

be based. Our algorithms are shown to exhibit a highly competitive performance, one which is comparable to (and sometimes better than) that of the state-of-the-art strategic decision-making algorithm in this domain.

In Section 2 we provide background on MCTS (Section 2.1) and the UCT tree policy (Section 2.2). Moreover, we describe the game "Diplomacy" (Section 2.3), the BANDANA framework [5] that we used for our Java implementation (Section 2.4) and briefly review related work that helped us with our project (Section 2.5). In Section 3 we present our approach and in Section 4 we detail our experiments and discuss our results. Section 5 draws conclusions and outlines future work directions.

2 BACKGROUND AND RELATED WORK

This section outlines the background theory that led to the development of Diplomacy agents using MCTS. The topics that will be discussed is the generic MCTS algorithm, the UCT method which is our selected tree policy, the basic rules of the Diplomacy multi-agent strategic game, and related work.

2.1 Monte Carlo Tree Search

The basic MCTS algorithm involves iteratively building a search tree until some predefined computational budget – typically a time, memory or iteration constraint – is reached. At this point the search is halted and the most promising (according to the information gathered by simulation) action is returned [1]. Each node in the search tree represents a state of the domain. Actions are represented by the directed links to the child nodes which represent the subsequent states. Four steps are applied per search iteration [3]:

(1) **Selection**: Starting at the root node, a child selection policy is recursively applied to descend through the tree until the most urgent expandable node is reached. A node is expandable if it represents a nonterminal state and has unexpanded children.
(2) **Expansion**: One (or more) child nodes are added to expand the tree, according to the available actions.
(3) **Simulation**:A simulation is run from the new node(s), according to the default policy, to produce an outcome.
(4) **Backpropagation**: The simulation result is backpropagated through the selected nodes to update their statistics.

The steps are summarized in pseudocode below [1]:

Algorithm 1: General MCTS approach

```
Function MctsSearch(s_0):
    create root node v_0 with state s_0;
    while within computational budget do
        v_i ← TreePolicy (v_0);
        Δ ← DefaultPolicy (s(v_0));
        Backup(v_i; Δ);
    return a(BestChild(v_0; 0));
```

The family of MCTS algorithms consists of a significant amount of variations of tree policies. Each of them tries to handle in an efficient way the dilemma between the exploitation of the currently gained knowledge about the actions and the exploration of the search space in order to improve that knowledge. In our project we use the most popular of them, the *Upper Confidence Bound for Trees (UCT)* algorithm [13].

2.2 The UTC Tree Policy

The UCT tree policy works as follows. As stated earlier, the key consideration is how to select a child node to expand. In UCT, a child node j is selected to maximize:

$$UCT = X_j + 2 * C_p * \sqrt{\frac{2 * \ln(n)}{Visits}}$$

where X_j is the normalized average reward ($0 \leq X_j \leq 1$), $C_p > 0$ and is a constant. The value $C_p = \frac{1}{\sqrt{2}}$ was shown by Kocsis and Szepesvari [13], to satisfy the Hoeffding inequality [10] with rewards in the range of [0, 1],n is the number of times the parent node has been visited, and Visits is the number of times current node has been visited. Algorithm 2 is pseudocode for the complete UCT Monte Carlo Tree Search algorithm [1]:

2.3 Diplomacy

Diplomacy is an American strategic board game created by Allan B. Calhamer in 1954 [2].In this section we present a brief overview of Diplomacy rules and gameplay. The game of Diplomacy is played by seven (7) players. Each player represents one of the seven "Great Powers of Europe" (Russia, Turkey, Austria, Italia, Great Britain, Germany, France) at the period before World War I. The map is divided into 75 Provinces. At the beginning of the game, each Power has 3 units, except Russia that has 4 units. Each unit has equal strength with every other unit. During gameplay, the units can support each other so they can increase their strength. There can be only one unit in a Province at a time. If a Power has one of its units to a Province after order resolution, we say that it controls that Province. 34 of the Provinces are Supply Centers (SCs). The first player that conquers 18 SCs is the winner of Diplomacy. Each round has a series of phases. The first round of the game is referred to as Spring 1901, followed by Fall 1901, Spring 1902, Fall 1902 etc. In each round all players simultaneously 'submit orders' for all of their units. Here are the phases in a complete two-turn year:
Spring

(1) Diplomatic phase
(2) Order writing phase

Algorithm 2: The UTC algorithm

```
1  Function UctSearch(s_0):
2      create root node v_0 with state s_0;
3      while within computational budget do
4          v_i ← TreePolicy (v_0);
5          Δ ←DefaultPolicy (s(v_0));
6          Backup(v_i; Δ);
7      return a(BestChild(v_0; 0));
8  Function TreePolicy(v):
9      while v is non-terminal do
10         if v not fully expanded then
11             return Expand(v);
12         else
13             v ←BestChild(v; C_p);
14         end
15     return v;
16 Function Expand(v):
17     choose α ∈ untried actions from A(s(v));
18     add a new child v' to v
19         with s(v') = f(s(v), α)
20         and a(v') = α;
21     return v';
22 Function BestChild(v, c):
23     return arg max_{v' children of v} Q(v')/N(v') + c√(2 ln N(v)/N(v'));
24 Function DefaultPolicy (s):
25     while s is non-terminal do
26         choose α ∈ A(s) uniformly at random;
27         s ← f(s, α);
28     return reward for state s;
29 Function Backup(v, Δ):
30     while v is not null do
31         N(v) ← N(u) + 1;
32         Q(v) ← Q(u) + Δ(v, p);
33         v ← parent of v
```

(3) Order resolution Phase
(4) Retreat and disbanding phase

Fall

(1) Diplomatic phase
(2) Order Writing phase
(3) Order Resolution Phase
(4) Retreat and Disbanding phase
(5) Gaining and Losing Units phase

During the *Diplomatic* phase, players are negotiating and making deals and agreements regarding their next moves. Alliances are made and strategies are set. "Conversations, deals, schemes and agreements will greatly affect the course of the game" [2].

During the *Order Writing* phase, every player has to give a set of orders, one for every unit it has on the map. These orders can be a HOLD order, a MOVETO order, or a SUPPORT order. (An illustration of such orders is shown in Figure 2.) The orders resolution process has complicated rules that will not be discussed here entirely [2]. In this section we provide only a short overview of these orders and their resolution process.

Figure 2: The first picture shows the possible moves of a unit from the English Channel. The second picture shows a unit from Gascony supporting a MOVE order of another unit from Marseilles to Burgundy [2].

Each MOVETO order will fail if its strength (1+1 for every SUPPORT order for it) is less than or equal to the strength of an enemy's HOLD or MOVETO order with the same destination. If both strengths are equal, then the enemy's MOVETO order fails as well (standoff). Each SUPPORT order will fail (cutoff) if the located Province is attacked by an enemy unit. Each HOLD order will fail if and only if there is a successful enemy's MOVETO order to the located Province. In any other case, the order is successfully resolved and the unit reaches its destination or supports a friendly unit. Every time a MOVETO order towards a Province that is the location of an enemy's SUPPORT/HOLD order or destination of an enemy's MOVETO order, succeeds, we say that we have a Dislodgement of that enemy unit.

During the *Retreat and Disbanding* phase, every player has to give a RETREAT order for every Dislodgement its Power has received. If there is no adjacent Region that the dislodged unit can retreat then the dislodged unit is removed from the map.

At the end of every Fall, the number of Supply Centers each Power owns is updated: If a SC is controlled by a unit, then it is added to the unit's Power. If a SC is not controlled by any unit then it remains to its previous owner. After we update the SCs lists, the number of the units on the map are counted for every Power. if a Power has more SCs than units, then it has to give BUILD orders, indicating the Home Province(s) for the new unit(s) to appear (Home Provinces are the starting Provinces of each Power). If a Power has more units than SCs, then it has to give REMOVE orders, indicating which unit(s) to be removed from the map.

2.4 The ANAC Diplomacy Challenge and the BANDANA Framework

The goal of the Automated Negotiation Agents Competition (ANAC) Diplomacy Challenge is the implementation of a negotiation algorithm on top of an existing Diplomacy playing agent [6]. That means that the strategic component of the participating Diplomacy playing agents is fixed and the same for all participants. The strategic component is the one of *D-Brane* [7], the strongest to date Diplomacy-playing agent.

The competition is run in the BANDANA framework [5]. BANDANA is a Java framework for the development of automated agents that play the game of Diplomacy. The BANDANA framework provides all the necessary Java classes to run a Diplomacy tournament and analyze the results. The main advantage of the BANDANA framework for our work here, is that it clearly distinguishes between the strategic module and the negotiation module of a Diplomacy agent. That offers us the chance to isolate the module of our interest, namely the strategic one, and focus on its evaluation while keeping all other factors the same.

2.5 Related Work

In this section we present some related work and experience that helped us either by offering ideas for the implementation of the Monte Carlo Tree Search algorithm, or by offering agents and algorithms to compete with.

2.5.1 Monte Carlo Tree Search in Strategic Games. Monte Carlo Tree Search algorithm is widely known for its application at Go. It has been used in conjunction with deep learning to build the successful and much-celebrated *AlhpaGo* algorithm [17]. Developed on October 2015, "*AlphaGo* is the first computer program that defeated a professional human Go player, defeated a Go world champion, and is considered the strongest Go player in history" [11].

Also, it has been used in various strategic games. We were inspired in our work here by its application in the non-deterministic game "Settlers of Catan", a multi-player board-turned-web-based game that necessitates strategic planning and negotiation skills [12, 16]. In particular, we found the MCTS implementation provided to us by the author of [12] to be extremely helpful.

The only other approach we are aware of that uses MCTS for the game of Diplomacy, was proposed in the 2016 Master's thesis of Xander Croes [4]. The thesis employed a very basic MCTS approach, with a search tree built with random sampling and without employing UCT. The approach was tested on a simplified board (only land provinces) and simple test maps with specific starting configurations (diamond, triangle, hexagon and square). Moreover, the approach was tested only against random players. This basic MCTS was shown to be doing better than random in all cases tested. However, when it was tested on the standard Diplomacy board, its performance was not convincing enough.

2.5.2 Diplomacy Agents. There are several agents that have been developed for the game of Diplomacy over the years. Here we describe three of them, which we later on use as opponents in order to evaluate our approach: an easy one, *RandomBot*; a moderate-strength one, *DumbBot* [15]; and the arguably strongest one found in the literature to date, *D-Brane* [7]. All of them are provided as executables by the BANDANA framework.

RandomBot. As the name indicates, *RandomBot* is an agent that just decides randomly which order to give to every unit. It also negotiates randomly. It makes random proposals and randomly approves or rejects the incoming proposals. It is also able to support friendly units when it is possible but it uses no tactics at all [5].

DumbBot. The *DumbBot* agent calculates a value for each region. Then, it makes a decision for an order for each unit, based on those values.
For each province, it calculates a value v as follows:

- If the agent possesses this SC: v = the size of the largest adjacent power.
- If the agent does not possess this SC: v = the size of the owning power.
- If it is not a SC: $v = 0$.

Then a table of Proximities is calculated as follows:

- $Proximity(0)$ for each coast, is the above value v, multiplied by a weighting factor.
- $Proximity(n)$ for each coast

$$Proximity_r(n) = \left(\frac{\sum_{x \in adj_{\text{Regions}(r)}} Proximity_x(n-1) + Proximity_r(n-1)}{5}\right) \cdot weightedFactor$$

Also, for each province, it calculates the number of adjacent units that the agent has (strength), and the number of adjacent units the enemies have (competition).

The *DumbBot* calculates an overall value for each region, based on all proximity values for that region, and the strength and competition for that region, each of which multiplied by a weight. For MOVE orders, the agent tries to move to the best adjacent region. However, there is a random chance that it moves to the second best, third best, etc. If the best place is where it already is, it holds. If the best place already has an occupying unit, or already has a unit moving there, it either supports that unit, or moves elsewhere, depending on whether the other unit is guaranteed to succeed or not.

D-Brane. The *D-Brane* agent uses a series of heuristics based on domain knowledge. We outline below the main algorithm of its strategic component [7]:

- A *deal* is a set of actions that the players agree to take. In the BANDANA framework, the agents are forced to perform the actions that are agreed in a deal. So, the deal is a restriction of an agent's set of possible actions. Each action that an agent takes, has to include the action indicated by the deal.
- Given a game state e (i.e. a configuration of units on the Diplomacy map), a deal x, and any player α_i the strategic component returns a set of orders for α_i for the game state, plus the number of Supply Centers that α_i is guaranteed to conquer if it submits those orders.
- A battle plan doe a player α_i to conquer a province p is "invinceble" if no opponent can choose any battle plan that would prevent α_i from conquering p. One has determined all battle plans for a given province p, for all players, it is easy to determine which of those battle plans are invincible. Similarly, one can determine the invincible pairs of battle plans. An "invincible pair" is a pair of battle plans (β_p, β_q) for two different Supply Centers p and q respectively, such that if α_1 plays both of these battle plans, then at least one of them is guaranteed to succeed.

- If e denotes some state of the game the set of all battle plans for player α_i to conquer or defend a Supply Center p is denoted $B^e_{i,p}$. The set of all Supply Centers is denoted SC.
- Now, given a game state e, an agreement x, and a player α_i, the strategic component of *D-Brane* works as follows:
  (1) For each $p \in SC$ determine all invincible plans from the set $B^e_{i,p}$.
  (2) For all pairs of Supply Centers $(p.q) \in SC \times SC$ (with $p \neq q$) determine all invincible pairs from the set $B^e_{i,p} \times B^e_{i,q}$.
  (3) Remove all invincible plans and invincible pairs that are not consistent with x.
  (4) Find the largest consistent combination of invincible plans and pairs, using And/Or tree search [8] with Branch & Bound.
  (5) For each province for which *D-Brane* has not been able to select an invincible plan or pair, select the strongest non-invincible battle plan that is consistent with x and the plans and pairs chosen in the previous step.
  (6) Return the full set of battle plans we have selected.

As shown in [7], *D-Brane* outperforms several previously developed Diplomacy agents, even when it is not engaging in negotiations. That is the reason why its strategic module has been adopted as the strategic module of choice for agents competing in the ANAC Diplomacy challenge (which focuses on the negotiations part of the game, as explained earlier).

3 OUR MONTE CARLO TREE SEARCH AGENTS

Our purpose in this work is to experiment exclusively with the strategic component of a Diplomacy agent. That means that our agent does not negotiate at all.

Our agent uses Monte Carlo Tree Search with UCT as its tree policy during the order writing phases of the game. As a heuristic, in each round it tries to maximize the number of Supply Centers (SCs) conquered during that round. The number of the owned SCs is the only reward that we take into account.

Each Tree Node contains the current game state (current layout of the units on the map), the joint action that led to that game state (apart from the root node), a list of joint actions that can be taken in order to expand the current node and create a new node, the reward value Δ (i.e. the number of SCs the Power owns), and the reward X_j that is expected to be found if the agent chooses the action that led to that state.

During the Backpropagation step, the Δ reward of the leaf node found at the end of the simulation is used at every parent node (until root node is reached) to update its X_j value that is used for the UCT formula in Section 2.2.

X_j is updated in the following manner:

$$X'_j = \frac{R + \Delta}{Visits} \tag{1}$$

where

- R is the previous cumulative number of SCs that has been owned until now;
- Δ is the number of SCs that are "found" at the leaf node at the end of the simulation step; and

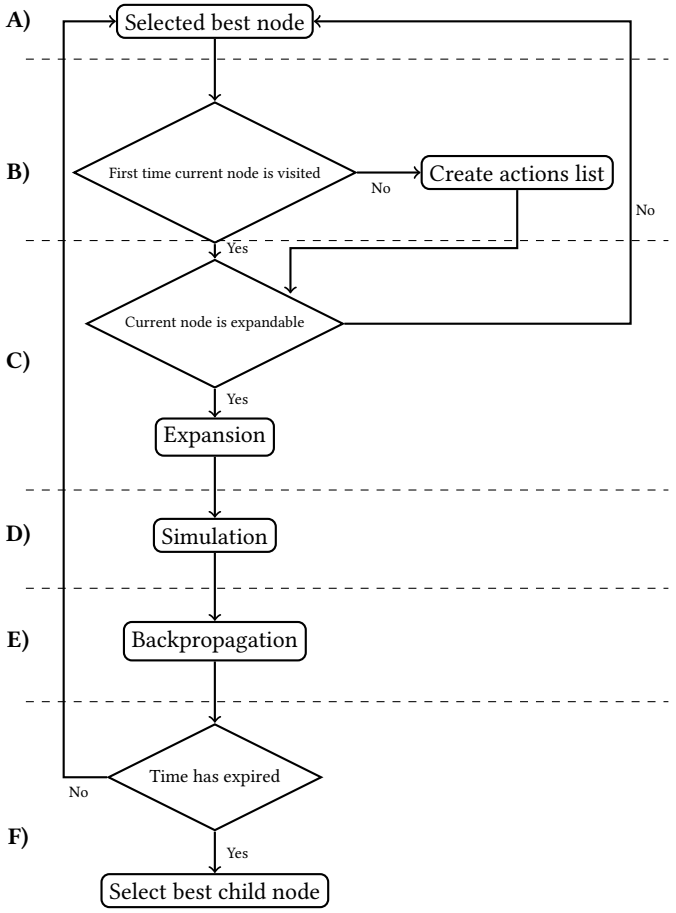

Figure 3: Flowchart of the MCTS_Agent algorithm

- $Visits$ is the number of times this node has been visited.

The *MCTS_Agent* algorithm can be summarized in the following steps (as can be seen at the flowchart of Figure 3):

A) Selection step using UCT tree policy.
B) The CreatePossibleActions function is called once for each selected node. It creates a list with all the accepted combinations of orders (actions) according to the pruning method that we apply.
C) Expansion step
D) Simulation step
E) Backpropagation step
F) Repeat the process for a specific amount of time—in our experiments we set this to 8 seconds. When the time expires, Select the child node with greater X_j value.

3.1 MCTS_Agent Version 1

The first *MCTS_Agent* version uses the following pruning method for step A: It gets all "accepted" orders for each unit using a heuristic rule.

- If the unit stands on a SC that we do not own, then return HOLD.

- If there is a HOLD or a MOVETO order with destination an adjacent SC then SUPPORT it.
- If an adjacent Province is SC that we do not own or control, then MOVE TO that SC.
- Else return every possible MOVETO and HOLD order.

Gets all the combinations of orders found. Calculates enemies' expected orders using a heuristic method:

- If the unit is next to a SC then expect a MOVETO order towards that SC else expect a HOLD order.

Creates a list with all possible actions from the current node.

3.2 MCTS_Agent Version 2

Version 2 of the *MCTS_Agent* is based upon Version 1 and also uses the following simulation policy at step D:
Instead of random simulation, we use a heuristic method:

- If the unit stands on a Province with a SC that we do not own yet, then HOLD.
- If there is a Province with SC next to a unit, then MOVETO that Province.
- If there is a MOVETO order already towards that Province then the unit is ordered to SUPPORT that order.
- Else the unit makes a random move.

3.3 MCTS_Agent Version 3

Version 3 of *MCTS_Agent* is based upon Version 2 and adds an extra feature at the simulation policy at step D. It locates SCs that are two steps away from the current location

3.4 MCTS_Agent Version 4

Version 4 of the *MCTS_Agent* is based upon Version 2. During simulation at step D, it calculates the expected enemy orders with the following heuristic method:

- If the enemy unit is next to an SC, then expect a MOVETO order towards that SC.
- Else expect a random order

3.5 MCTS_Agent Version 5

Version 5 of the *MCTS_Agent* is based upon Version 4. The main idea of this version is to find a way to guide MCTS in order to expand the nodes using the most promising actions first. For that reason, *MCTS_Agent* creates an a priori evaluation of the action of each node. For all the previous versions, after a node was selected, the action needed for the node's expansion was chosen randomly. For version 5 of the *MCTS_Agent* , at the expansion step (step C), the node is expanded using the action with the maximum value.

This evaluation is made through a novel weighting system we devised, and which is based on the heuristic DumbBot algorithm for move selection. It calculates a weight for each map Region, and then calculates each action's value based on those weights.
For each Region, it calculates v as follows:

- If it is a SC owned by MCTS_Agent: v = the size of the adjacent enemies' units
- If it is a SC not owned by MCTS_Agent: v = a constant (i.e. 6) minus the size of the owning power
- If it is not a SC: $v = 0$.

For each Region r, the proximity values are calculated first:

$$Proximity_r(0) = v * proximityWeight(0) \tag{2}$$

$$Proximity_r(n) = \Big(\frac{\sum_{x \in adj_{\text{Regions}(r)}} Proximity_x(n-1)}{5} + Proximity_r(n-1)\Big) proximityWeight(n) \tag{3}$$

and give rise to a regional weight:

$$Weight(r) = \sum_{n=0}^{9} Proximity_r(n) \tag{4}$$

Then the value of each action a is calculated as:

$$Value(a) = \sum_{r \in R_a} Weight(r) \tag{5}$$

where R_a refers to the set of a's destination regions (e.g., regions on which a moves, holds or supports a specified number of units). At the expansion step, we always choose the action with the highest value:

$$\max_{a \in \text{listOfActions}} Value(a) \tag{6}$$

This version of the agent has the need of extra pruning. So, if an order has for destination a region with value less than the mean value of all regions, then the order is pruned.

Note that *MCTS_Agent* version 5 has the same simulation policy as *MCTS_Agent* version 4.

3.6 MCTS_Agent Version 6

Version 6 of the *MCTS_Agent* is based upon Version 5. It tries a different proportion between the time spent for expansion and the time spent for simulation, in favor of the first. That happens by adding a second expansion step, before the simulation.

3.7 MCTS_Agent Version 7

Version 7 of the *MCTS_Agent* is based upon Version 5. It uses the weighting system of Version 5 in its simulation policy (step D). In each simulation, it chooses the highest-valued action a:

$$\max_{a \in \text{listOfActions}} Value(a)$$

where $Value$ is defined as in Equations (2)-(5) (see Section 3.5.)

3.8 MCTS_Agent Version 8

Version 8 of the *MCTS_Agent* is based upon Version 5. At simulation step (step D), the agent plays against a variety of opponents trying to achieve a better exploration of the search space [14]. Thus, for this version, each opponent during the simulation is chosen randomly between three options: (A) Random Bot, (B) a heuristic using the weighting system as *MCTS_Agent* Version 7 does (see Section 3.7), (C) a simple and greedy heuristic: If the unit is next to a SC then expect a MOVETO order towards that SC else expect a HOLD order.

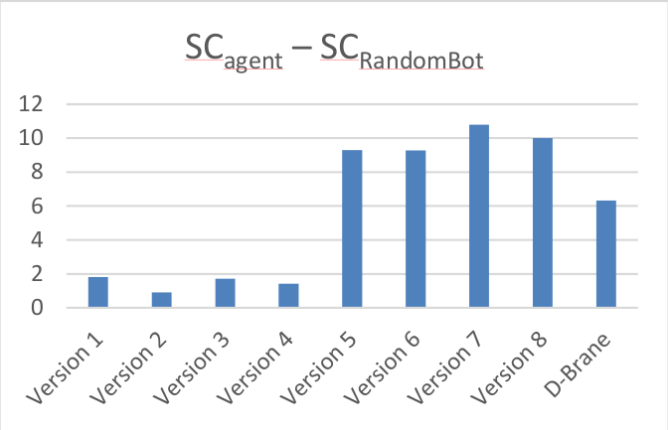

Figure 4: Versions of the *MCTS_Agent* and *D-Brane* vs *RandomBot*. Average of 30 games.

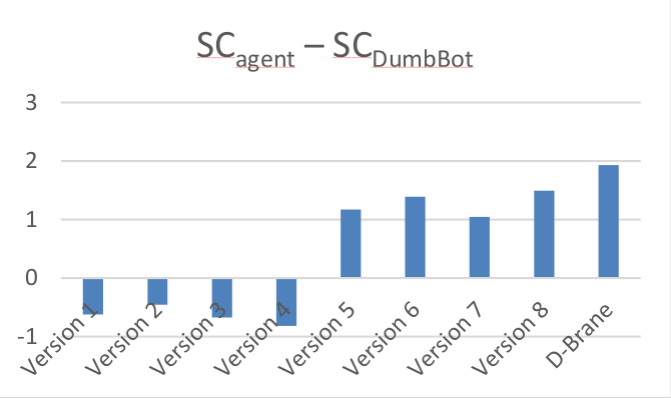

Figure 5: Versions of the *MCTS_Agent* and *D-Brane* vs *DumbBot* . Average of 30 games.

4 EXPERIMENTAL EVALUATION

In this section we present a systematic comparison and evaluation of our MCTS agents, using *RandomBot*, *DumbBot* and *D-Brane* as their opponents. *MCTS Agents*, *D-Brane* and *DumbBot* are non-negotiating agents. The only negotiating agent is RandomBot that negotiates randomly with other *RandomBots*. Each game in our experiments lasts for 5 years (though in later experiments we alter that value to 10, as explained below).

4.1 MCTS Agents against Random Bots

We begin by evaluating the performance of our agents, as well as that of *D-Brane* , against *RandomBots*. Each agent in Figure 4 plays a 30-game tournament against 6 *RandomBots*.

Each of the values shown on the bars in Figure 4 corresponds to the value of the following metric:

$$SC_{\text{agent}} - SC_{\text{RandomBot}},$$

where SC_{agent} is the average number of Supply Centers that the agent (*MCTS_Agent* or *D-Brane*) owns at a game and $SC_{\text{RandomBot}}$ is the average number of SCs that *RandomBot* owns in a game (when there are more than one *RandomBots* we take the mean).

Firstly, we see that versions 1, 2, 3 and 4 of the *MCTS_Agent* outplay *RandomBot* but are not comparable with *D-Brane* . They only conquer the Supply Centers that are near to their starting positions and that is enough to outplay *RandomBot*. That happens because of the greedy heuristic used as a pruning method. MCTS does not help the agent to behave strategically in the long run.

On the other hand, we observe that versions 5, 6, 7 and 8 of our *MCTS_Agent* expand their territories and conquer Supply Centers at a much better rate than *D-Brane* when playing against Random Bots. It is also important to highlight that they achieved some solo victories despite the strict year limit. In these versions, the algorithm used for sorting each node's actions list (described in Section 3.5) improved the effectiveness of MCTS substantially.

4.2 MCTS Agents against DumbBots

In this subsection we evaluate the performance of our agents (and also that of *D-Brane*) against *DumbBots*. Each agent in Figure 5 plays a 30-game tournament against 6 *DumbBots*. Each of the values shown on the bars in Figure5 corresponds to the value of the following metric:

$$SC_{\text{agent}} - SC_{\text{DumbBot}},$$

where SCagent is the average number of Supply Centers that the agent (*MCTS_Agent* or *D-Brane*) owns at a game and SC_{DumbBot} is the average number of SCs that *DumbBot* owns in a game (when there are more than one *DumbBots* we take the mean).

One observation is that the first 4 *MCTS_Agent* versions are outplayed by *DumbBot* . However, we see that versions 5, 6, 7 and 8 clearly outperform *DumbBot* , but not as much as *D-Brane* does. Despite its name, *DumbBot* is a competitive agent, so it is natural that we do not observe the same outstanding performance we witnessed when MCTS agents played against *RandomBots*.

4.3 MCTS Agents against D-Brane

In this subsection we provide a thorough evaluation of our approach against *D-Brane* . Each agent in this setting played three 30-game tournaments against one, three and five *D-Branes* respectively. We chose this method of evaluation because it is important in a 7-player game to consider the quality of the nearby opponents. There is now a greater need for a better strategy in order to expand when having several tough (*D-Brane*) neighbors, than when having *RandomBots* or *DumbBots* as opponents.

The tournaments (the three columns of the table and corresponding bars in Figure 6) are between the following agents:

- 1^{st}: 1 *MCTS_Agent* , 1 *D-Brane* , 2 *DumbBots*, 3 *RandomBots*.
- 2^{nd}: 1 *MCTS_Agent* , 3 *D-Brane* , 2 *DumbBots*, 1 *RandomBots*.
- 3^{rd}: 1 *MCTS_Agent* , 5 *D-Brane* , 1 *DumbBots*, 0 *RandomBots*.

Each value in Figure 6 represents the subtraction:

$$SC_{\text{MCTS}} - SC_{\text{DBrane}},$$

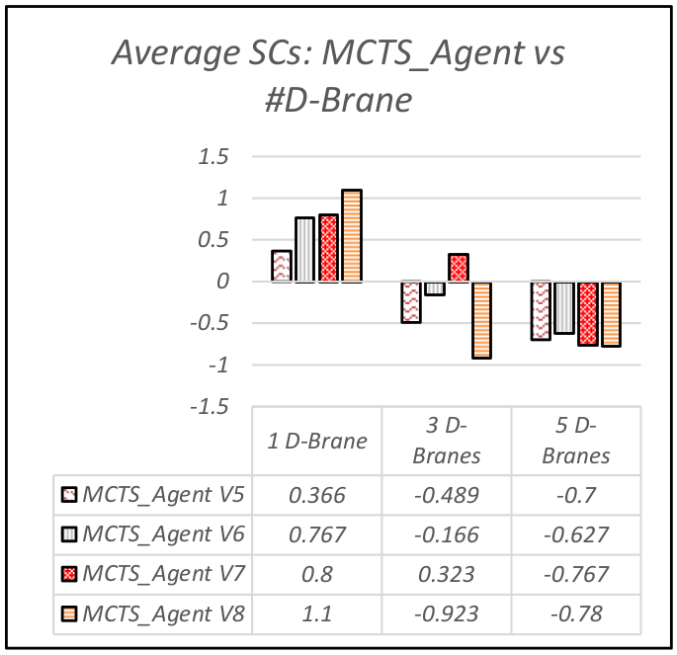

Figure 6: One *MCTS_Agent* vs one, three or five *D-Brane* agents. Subtraction of the average SCs owned by the agents at every tournament. Average of 30 games.

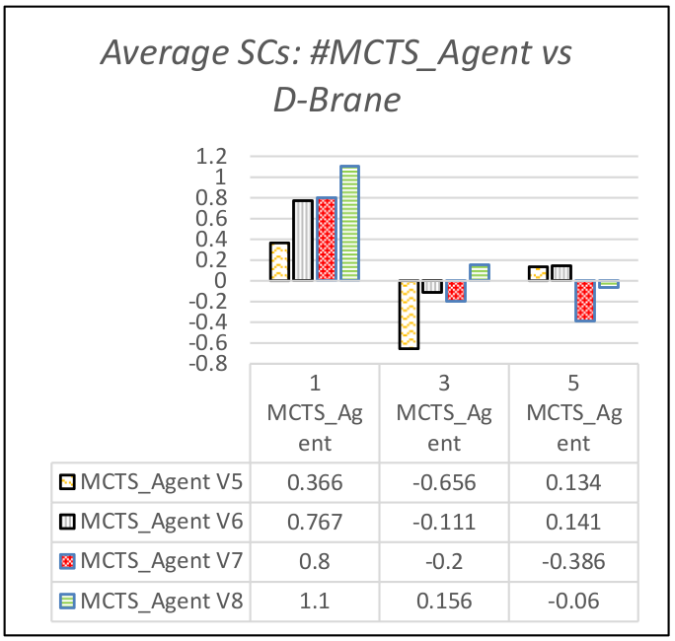

Figure 7: One *D-Brane* agent vs one, three or five *MCTS_Agents*. Subtraction of the average SCs owned by the agents at every tournament. Average over 30 games.

Table 1: Version 8 of *MCTS_Agent* vs *D-Brane* . 10 years limit. Average of 20 games.

	Supply Centers (Avg)	Solo Victories (Avg)	Average Rank (of each specific competitor)
MCTS_Agent	15.0	50%	1.375
D-Brane	11.3	25%	1.925
RandomBot	1.49 ± 0.64	0	4.94 ± 0.46

where SC_{MCTS} is the average number of Supply Centers that *MCTS_Agent* owns at a game and SC_{DBrane} is the average number of Supply Centers that *D-Brane* owns at a game (when there are more than one *D-Brane* agents we take the mean value).

We observe that all *MCTS_Agents* beat *D-Brane* at one vs one tournaments, but they lose at the 1 vs 3 and 1 vs 5 tournaments (with the exception of *MCTS_AgentV7* vs 3 *D-Branes*).

In Figure 7 we present more comparative results for *MCTS_Agents* version 5, 6, 7 and 8. A *D-Brane* agent has played a 30-game tournament against one, three and five agents of each version of *MCTS_Agent* .

The tournaments (the three columns of the graph) are between the following agents:

- 1st: 1 *MCTS_Agent* , 1 *D-Brane* , 2 *DumbBots*, 3 *RandomBots*.
- 2nd: 3 *MCTS_Agent* , 1 *D-Brane* , 2 *DumbBots*, 1 *RandomBots*.
- 3rd: 5 *MCTS_Agent* , 1 *D-Brane* , 1 *DumbBots*, 0 *RandomBots*.

Each value of the graph represents the subtraction:

$$SC_{\text{MCTS}} - SC_{\text{DBrane}},$$

where SC_{MCTS} is the average number of Supply Centers that *MCTS_Agent* owns at a game (when there are more than one *MCTS_Agents* we take the mean value) and SC_{DBrane} is the average number of Supply Centers that *D-Brane* owns at a game.

First, we observe that all (v5, v6, v7 and v8) *MCTS_Agent* versions beat *D-Brane* in one vs one tournaments. At 3 *MCTS_Agents* vs 1 *D-Brane* tournaments we observe that *MCTS_AgentV8* slightly outperforms *D-Brane* . Also, at 5 *MCTS_Agent* vs 1 *D-Brane* tournaments we observe that versions 5 and 6 of the *MCTS_Agent* slightly outplay *D-Brane* .

We then conducted an additional experiment involving one of our most promising versions of our *MCTS_Agents* facing one *D-Brane* and five *RandomBots*., increasing the years limit for the game to 10. In Table 1 we see that our *MCTS_Agent* v8 clearly outperforms *D-Brane* in all metrics. Interestingly, it achieves many Solo Victories. That confirms that, at least when the rest of the opponents are weak (for example, *RandomBots*.), the *MCTS_Agent* conquests SCs in a faster way than *D-Brane* does.

As was to be expected, however, when there are many strong *D-Brane* opponents in the game, *MCTS_Agent* cannot dominate anymore, and in fact it loses to *D-Brane* (see Table 2). Regardless, it is still a competitive agent, achieving an average rank of approximately 4, with its five *D-Brane* competitors achieving an average

Table 2: Version 8 of *MCTS_Agent* vs 5 *D-Brane* . 10 years limit. Average of 20 games.

	Supply Centers (Avg)	Solo Victories (Avg)	Average Rank (of each specific competitor)
D-Brane	5.91 ± 0.31	1%	3.41 ± 0.265
MCTS_Agent	4.15	0	4.025
RandomBot	0.3	0	6.925

rank of 3.41. Essentially, the *MCTS_Agent* and the five *D-Branes* split the 34 Supply Centers almost equally among themselves.

4.4 Discussion of the Results

Our experiments show that *MCTS_Agent* is a competitive and comparable agent to *D-Brane* . In certain settings, it can outplay *D-Brane* (for example at one vs one tournaments) and in the settings that it is outplayed, it still exhibits a competitive performance.

Now, the first challenge we had to face while implementing MCTS in this domain, is the urgent need of heuristics. The search space is huge and it is impossible for the agent to simulate enough to obtain reasonable rewards. Even if we had had unlimited time to make a decision, it would have been computationally impossible for the agent to even count all the possible actions that the algorithm could make from a state that our Power has the control of many units (e.g. more than ten). Arguably, this is why we got much better results when we used the weighting system heuristic (in versions 5, 6, 7 and 8): we now were able to sort the action lists, and thus indicate to the agent which moves/actions should be expanded and searched first.

For every state, the agent estimates the orders that the enemy units will make. For this purpose, we use a heuristic (see Section 3.1). It is not efficient to suppose that the enemies are using a competitive algorithm (e.g. *D-Brane*) because running that algorithm for six times (the number of enemies) would cost a great amount of time. So, we chose a rather greedy heuristic in order to make this estimation. This saves us time that is used for more iterations of the algorithm, but makes the agent exhibit a poor defensive behavior: it hardly ever recognizes a threat unless it is too close.

The first four versions of the agent (version 1, 2, 3 and 4) are not competitive. They are outperformed by both *D-Brane* and by *DumbBot* . The MCTS algorithm does not seem to find any clever move to make, and the agent manages to occupy just the few Supply Centers that are located at the adjacent Provinces of its starting position. We believe that these results are mainly affected by the pruning method used.

Implementing a weighting system method for sorting each node's action list, and thus effectively injecting more domain knowledge to the agent, allows MCTS to produce much better results, because the weights indicate which actions should be expanded first, i.e. helps the agent focus on the most promising actions. In our view, this is the main reason why *MCTS_Agent* 's versions 5, 6, 7 and 8 outperform *DumbBot* .

MCTS_Agent outperformed *D-Brane* when playing one vs one tournaments and was slightly outperformed in the other settings (see Figures 6 and 7). It is obvious from our results that *MCTS_Agent* conquests SCs in a faster way when facing *RandomBots* than when facing stronger opponents. This rate deteriorates somewhat when facing *DumbBots* (see Figure 5), and deteriorates further when facing *D-Branes* (see Figure 6). In general, it is obvious from our results that, though an *MCTS_Agent* cannot be a clear winner when facing many strong opponents (regardless of whether these are *D-Branes* or other *MCTS_agents*), it is always a very competitive opponent.

The experiments in which we set the year limit to ten years (see Tables 1 and 2) confirm the previous conclusions. In the one vs one tournament, the *MCTS_Agent* outperforms *D-Brane* (captures more Supply Centers and achieves more Solo Victories). In the one vs five tournament, *MCTS_Agent* is outperformed by *D-Brane* but still it maintains a competitive performance.

While comparing the MCTS agents, we notice that version 8 has a slight advantage against the others. However, this advantage does not exceed a value that could let us conclude that it is definitely better than the others (in a statistically significant manner). Notice also that this advantage was not maintained when it faced three *D-Brane* agents (see Figure 6). Perhaps the safest conclusion we can reach regarding the relative ranking of the MCTS variants, is that Versions 8 and 7 appear to be the most successful *MCTS_Agent* ones, with versions 5 and 6 coming close.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we adapted and systematically evaluated Monte Carlo Tree Search for the Diplomacy game domain. We put forward 8 different MCTS variants that incorporate levels of domain knowledge, and devised novel heuristics to aid this task. As such, our work resulted to the creation of a basic framework for further research on the use of MCTS in Diplomacy

In our experiments, we compare the performance of our agents against several opponents, including the state-of-the-art Diplomacy agent, *D-Brane* . Our results show that *MCTS_Agent* outperforms *D-Brane* in certain settings (see specifically in one vs one settings as reported in Chapter 4), and in most other cases has a competitive performance, comparable to that of *D-Brane* .

The success of the *MCTS_Agents* v5, 6, 7 and 8 which incorporate the heuristic sorting of the actions' lists technique, demonstrates that injecting high quality domain knowledge (especially in such a complicated domain) improves MCTS performance.

In future work, we intend to experiment further with altering the year limit and the time limit of the games. Making the approach adaptive to changing such limits, is also an interesting research direction. Furthermore, we intend to experiment with different MCTS tree policy enhancement methods such as Bayesian UCT [18] or AMAF [9], in anticipation of an improvement in performance, as implied by their use in other domains.

ACKNOWLEDGMENTS

We kindly thank Athina Georgara for invaluable help with shaping the final version of this paper.

REFERENCES

[1] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Stephen Tavener, Diego Perez, Spyridon Samothrakis, Simon Colton, and et al. 2012. A survey of Monte Carlo tree search methods. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI* (2012).

[2] Alan B. Calhamer. 2000. The Rules of Diplomacy, 4th Edition, The Avalon Hill Games Co. http://www.diplomacy-archive.com/resources/rulebooks/2000AH4th.pdf

[3] Guillaume Chaslot, Sander Bakkes, István Szita, and Pieter Spronck. 2008. Monte-Carlo Tree Search: A New Framework for Game AI. In *AIIDE*.

[4] Xander Croes. 2016. *Tree Search Methods for Diplomacy Agents.* Master's thesis. Universiteit Leiden. https://theses.liacs.nl/pdf/xandercroesmaster2016.pdf

[5] Dave de Jonge. 2019. The BANDANA Framework v1.3. https://www.iiia.csic.es/~davedejonge/bandana/files/Bandana1.3Manual.pdf

[6] Dave de Jonge, Tim Baarslag, Reyhan Aydoğan, Catholijn Jonker, Katsuhide Fujita, and Takayuki Ito. 2019. The Challenge of Negotiation in the Game of Diplomacy. In *Agreement Technologies 2018, Revised Selected Papers*, Marin Lujak (Ed.). Springer International Publishing, Springer International Publishing, Cham, 100–114.

[7] Dave de Jonge and Carles Sierra. 2017. D-Brane: a diplomacy playing agent for automated negotiations research. *Applied Intelligence* 47 (02/2017 2017), 158–177.

[8] Rina Dechter and Robert Mateescu. 2007. AND/OR search spaces for graphical models. *Artificial Intelligence* 171, 2 (2007), 73 – 106. https://doi.org/10.1016/j.artint.2006.11.003

[9] David P. Helmbold and Aleatha Parker-Wood. 2009. All-Moves-As-First Heuristics in Monte-Carlo Go. In *Proceedings of the 2009 International Conference on Artificial Intelligence (IC-AI) (ICAI'09).* 605–610.

[10] Wassily Hoeffding. 1994. *Probability Inequalities for sums of Bounded Random Variables.* Springer New York, New York, NY, 409–426. https://doi.org/10.1007/978-1-4612-0865-5_26

[11] Sean D Holcomb, Shaun V Ault, William K Porter, Guifen Mao, and Jin Wang. 2018. Overview on DeepMind and Its AlphaGo Zero AI. In *ICBDE '18: Proceedings of the 2018 International Conference on Big Data and Education* (Honolulu, HI, USA). Association for Computing Machinery, New York, NY, USA.

[12] Emanouil Karamalegos. 2016. *Monte Carlo Tree Search in the "Settlers of Catan" Strategy Game.* Master's thesis. Technical University of Crete. https://doi.org/10.26233/heallink.tuc.66891

[13] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML 2006*, Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 282–293.

[14] Leandro Soriano Marcolino and Hitoshi Matsubara. 2011. Multi-Agent Monte Carlo Go. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1* (Taipei, Taiwan) *(AAMAS '11).* International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 21–28.

[15] David Norman. 2003. David's Diplomacy AI Page. http://www.ellought.demon.co.uk/dipai

[16] Konstantinos P. Panousis. 2014. *Real-time Planning and Learning in the "Settlers of Catan".* Master's thesis. Technical University of Crete. https://doi.org/10.26233/heallink.tuc.18113

[17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (01 Oct 2017), 354–359. https://doi.org/10.1038/nature24270

[18] Gerald Tesauro, V T Rajan, and Richard Segal. 2010. Bayesian Inference in Monte-Carlo Tree Search. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence* (Catalina Island, CA) *(UAI'10).* AUAI Press, Arlington, Virginia, USA, 580–588.