

# Towards Understanding the Effects of Evolving the MCTS UCT Selection Policy

Fred Valdez Ameneiro and Edgar Galván\*

**Abstract**—Monte Carlo Tree Search (MCTS) is a sampling best-first method to search for optimal decisions. The success of MCTS depends heavily on how the MCTS statistical tree is built and the selection policy plays a fundamental role in this. A particular selection policy that works particularly well, widely adopted in MCTS, is the Upper Confidence Bounds for Trees, referred to as UCT. Other more sophisticated bounds have been proposed by the community with the goal to improve MCTS’s performance on particular problems. Thus, it is evident that while the MCTS UCT behaves generally well, some variants might behave better. As a result of this, multiple works have been proposed to evolve a selection policy to be used in MCTS. Although all these works are inspiring, none of them have carried out an in-depth analysis shedding light under what circumstances an evolved alternative of MCTS UCT might be beneficial in MCTS due to focusing on a single type of problem. In sharp contrast to this, in this work we use five functions of different nature, going from a unimodal function, covering multimodal functions to deceptive functions. We demonstrate how the evolution of the MCTS UCT might be beneficial in multimodal and deceptive scenarios, whereas the MCTS UCT is robust in unimodal scenarios and competitive in the rest of the scenarios used in this study.

**Index Terms**—Monte Carlo Tree Search, UCT.

## I. INTRODUCTION

Monte Carlo Tree Search (MCTS) is a sampling method for finding *optimal decisions* by performing random samples in the decision space and building a tree according to partial results. The evaluation function of MCTS relies directly on the simulation’s outcomes. The optimal search tree is guaranteed to be found with infinite memory and computation [22]. However, in more realistic scenarios, MCTS can produce good approximate solutions [12], [13].

MCTS has gained popularity in two-player board games partly thanks to its success in the game of Go [28], including beating professional human players. The diversification of MCTS in other research areas is extensive. For instance, MCTS has been explored in energy-based problems [12], [13] and in the design of deep neural network (DNN) architectures [31]. These two extreme examples, along with the applicability in games [16], [30], demonstrate the successful versatility, use and applicability of MCTS in different problem domains.

The success or failure of MCTS depends heavily on how the MCTS statistical tree is built. The selection policy, responsible

for this, behaves well when using the Upper Confidence Bounds for Trees (UCT) [23]. Some conditions are to be met for this to work well, for example, the selection of a child of a given node is based on the exploration/exploitation trade-off. To this end, the UCT expression is normally used, yielding good results. Some of the parameters’ values can be changed contributing to a better performing MCTS. Moreover, some sophisticated bounds have been proposed such as the single-player MCTS, adding a third term to the UCT formula and changing the value of a parameter [27]. The UCB1-tuned also modifies the UCT expression to reduce the impact of the exploration term [2]. Thus, it is evident that while the UCT performs well on a range of problems, its adjustment or modification can have a more positive effect.

It is clear then to see that the MCTS’s selection policy when adjusted properly can have a more positive impact on a given problem. Motivated by this, we use Evolutionary Algorithms (EAs) [9] to evolve the UCT formula. Specifically, we use a semantic-based approach to do so inspired by the positive impact that semantics has in Genetic Programming [24] as shown by multiple works [17], [11], [29] including our studies in Multi-Objective GP [18], [14].

There are some interesting works using EAs in MCTS. Cazenave [8] used GP to evolve the UCT formula to be used in MCTS. The author used Swiss tournament selection, reproduction and mutation to be applied in the 128 or 256 individuals. The author tested his approach using the game of Go. He demonstrated how his GP approach outperformed MCTS and RAVE when a relatively small number of play outs were used, but UCT outperformed his approach when more play outs were allowed.

Motivated by Cazenave’s studies, a decade later, Bravi et al. [5] evolved the MCTS UCB1 formula tested in the General Video Game AI framework. They used a population of 100 GP trees and the initial population started with a seeded UCT individual and 99 random trees. In their studies, they considered three different scenarios: (a) only given access to the same information as UCB1, (b) given access to additional game-independent information, and (c) given access to game-specific information. The authors showed that, on average, Scenario (c) outperformed the rest of the scenarios including the UCB1.

Holmgård et al. [20] used GP to evolve persona-specific evaluation formulae to be used in MCTS instead of the UCB1 in the deterministic game of MiniDungeons 2. To evolve the 100 GP individuals over 100 generations, the authors used selection, crossover, mutation and elitism, as well as an island model. They reported that their evolved personas were able to

\*Leading and corresponding author.

Fred Valdez Ameneiro and Edgar Galván are both with the Naturally Inspired Computation Research Group and with the Department of Computer Science, Maynooth University, Lero, Ireland e-mails: fred.valdezameneiro.2019@mumail.ie and edgar.galvan@mu.ie

play the game more efficiently compared to the UCB1 agents.

Other interesting works include those carried out by Alhejali and Lucas [1], where they used a GP system to enhance MCTS during the roll out simulations tested in the game of Ms PacMan. The authors used multiple functions including comparison, conditional, logical operators. They used 100 or 500 individuals evolved, using selection and mutation operators, over 50 or 100 generations, where a single run took around 18 days to finish, for the latter case.

Another interesting work is related to handling prohibitive branch factors in MCTS through EAs, as proposed by Baier and Cowling [4] in the deterministic game of Hero Academy. Lucas et al. [25] used an EA as a source of control parameters to bias the roll outs of MCTS. They showed how their proposed approach significantly outperforms the vanilla MCTS using the Mountain Car benchmark problem and a simplified version of Space Invaders.

These works are inspiring, but the contributions of this work are novel and timely.

- 1) Firstly, instead of using a single type of problem as normally done so far by the community, we use five functions of different nature and complexity, going from a unimodal function, covering multimodal functions, to deceptive functions.
- 2) Secondly, we use a state-of-the-art EA algorithm, inspired by semantics, to evolve a selection policy to be used instead of the MCTS UCT.
- 3) Thirdly, instead of using a large population size and a large number of generations as done in all the previous works described before, we use a small population size evolved by a few generations to see if it is possible to successfully evolve a selection policy to be used in lieu of the MCTS UCT.
- 4) Fourthly, we compare five different variants of the MCTS UCT vs. our semantic-inspired EA and shed some light under what circumstances an evolved selection policy might have a positive effect on MCTS.

The rest of this paper is organised as follows. Section II provides some background in MCTS, EAs, semantics and in the functions used in this work. Section III discusses in detail the controllers used in this work. Section IV presents the experimental setup. Section V discusses the results obtained by each of the controllers. Section VI draws some conclusions.

## II. BACKGROUND

### A. The Mechanics Behind MCTS

MCTS relies on two key elements: (a) that the true value of an action can be approximated using simulations, and (b) that these values can be used to adjust the policy towards a best-first strategy. The algorithm builds a partial tree, guided by the results of previous exploration of that tree. Thus, the algorithm iteratively builds a tree until a condition is reached or satisfied (e.g., number of simulations, time given to Monte Carlo simulations), then the search is halted and the best performing action is executed. In the tree, each node represents a state, and directed links to child nodes represents actions leading to subsequent states. Like many AI techniques, MCTS

has several variants. Perhaps, the most accepted steps involved in MCTS are those described in [6] and are the following: (a) *Selection*: a selection policy is recursively applied to descend through the built tree until an expandable (a node is classified as expandable if it represents a non-terminal state and also, if it has unvisited child nodes) node has been reached, (b) *Expansion*: normally one child is added to expand the tree subject to available actions, (c) *Simulation*: from the new added nodes, a simulation is run to get an outcome (e.g., reward value), and (d) *Back-propagation*: the outcome obtained from the simulation step is back-propagated through the selected nodes to update their corresponding statistics.

Simulations in MCTS start from the root state (e.g., actual position) and are divided in two stages: when the state is added in the tree, a tree policy is used to select the actions (the selection step is a key element and it is discussed in detail later in this section). A default policy is used to roll out simulations to completion, otherwise.

One element that contributed to enhance the efficiency in MCTS was the selection mechanism proposed in [23]. The main idea of the proposed selection mechanism was to design a Monte Carlo search algorithm that had a small probability error if stopped prematurely and that converged to the optimal solution given enough time. That is, a selection mechanism that nicely balances exploration vs. exploitation, explained in the following paragraphs.

### B. Upper Confidence Bounds for Trees

As indicated previously, MCTS works by approximating ‘real’ values of the actions that may be taken from the current state. This is achieved through building a search or decision tree. The success of MCTS depends heavily on how the tree is built and the selection process plays a fundamental role in this. One particular selection mechanism that has proven to be reliable is the UCB1 tree policy [23]. Formally, UCB1 is defined as:

$$UCT = \bar{X}_j + C \sqrt{\frac{2 \cdot \ln \cdot n}{n_j}} \quad (1)$$

where  $n$  is the number of times the parent node has been visited,  $n_j$  is the number of times child  $j$  has been visited and  $K > 0$  is a constant. In case of a tie for selecting a child node, a random selection is normally used [23].

Thus, this selection mechanism works due to its emphasis on balancing both exploitation (first part of Eq. 1) and exploration (second part of Eq. 1).

### C. Evolutionary Algorithms

Evolutionary Algorithms (EAs) [3], [9], also known as Evolutionary Computation systems, refer to a set of stochastic optimisation bio-inspired algorithms that use evolutionary principles to build robust adaptive systems. The key element to these algorithms is undoubtedly flexibility in allowing the practitioner to use elements from two or more different EAs techniques. Consequently, the boundaries between these approaches are no longer distinct allowing a more holistic EA

framework to emerge, such as the one adopted in this research. EAs work with a population of  $\mu$ -encoded (representation of the) potential solutions to a particular problem. Each potential solution, commonly known as an individual, represents a point in the search space, where the optimal solution lies. The population is evolved by means of genetic operators, over a number of generations, to produce better results to the problem. Each individual is evaluated using a fitness function to determine how good or bad the individual is for the problem at hand. The fitness value assigned to each individual in the population probabilistically determines how successful the individual will be at propagating (part of) its code to future generations.

The evolutionary process is carried out by using genetic operators. Selection, crossover and mutation are the key operators used in most EAs. The selection operator is in charge of choosing one or more individuals from the population based on their fitness values. Multiple selection operators have been proposed. One of the most popular selection operators is tournament selection where the best individual is selected from a pool, normally of size =  $[2 - 7]$ , from the population. The stochastic crossover, also known as recombination, operator exchanges material normally from two selected individuals. This operator is in charge of exploiting the search space. The stochastic mutation operator makes random changes to the genes of the individual and is in charge of exploring the search space. The mutation operator is important to guarantee diversity in the population as well as recovering genetic material lost during evolution. This evolutionary process is repeated until a stopping condition is reached such as until a maximum number of generations has been executed. The population, at this stage, contains the best evolved potential solutions to the problem and may also represent the global optimal solution.

The field has its origins in four landmark evolutionary methods: Genetic Algorithms [19], Evolution Strategies [26], Evolutionary Programming [10] and Genetic Programming [24]. In this work we briefly describe the two methods employed in this work: Evolution Strategies and Genetic Programming. The second author's work in Neuroevolution in Deep Neural Networks [15] provides a nice summary of all of these EAs.

1) *Evolutionary Algorithm: Genetic Programming (GP)*: This EA was popularised by Koza [24]. GP is a form of automated programming where individuals are randomly created by using functional and terminal sets required to solve a given problem. Multiple types of GP have been proposed in the literature with the typical tree-like structure being the predominant form of GP in EAs.

2) *Evolutionary Algorithm: Evolution Strategies (ES)*: These EAs were introduced in the 1960s by Rechenberg [26]. ES are generally applied to real-valued representations of optimisation problems. In ES, mutation is the main operator whereas crossover is the secondary, optional, operator. Historically, there were two basic forms of ES, known as the  $(\mu, \lambda)$ -ES and the  $(\mu + \lambda)$ -ES.  $\mu$  refers to the size of the parent population, whereas  $\lambda$  refers to the number of offspring that are produced in the following generation before selection is applied. In the former ES, the offspring replace the parents

whereas in the latter form of ES, selection is applied to both offspring and parents to form the population in the following generation.

#### D. Semantics

For clarity purposes, we first briefly give some definitions on semantics, based on the first author's work [11], that will allow us to describe our approach later in Section III.

Let  $p \in P$  be a program from a language  $P$ . When  $p$  is applied to an input  $in \in I$ ,  $p$  produces an output  $p(in)$ .

*Def. 1:* The semantic mapping function  $s : P \rightarrow S$  maps any program  $g$  to its semantics  $s(g)$ .

This means,  $s(g_1) = s(g_2) \iff \forall in \in I : g_1(in) = g_2(in)$ . The semantics specified in Def. 1 has three properties. Firstly, every program has only and only one semantics. Secondly, two or more programs can have the same semantics. Thirdly, programs that produce different outputs have different semantics. Def. 1 is general as it does not specify how semantics is represented. This work is inspired by a popular version of semantics GP where the semantics of a program is defined as the vector of output values computed by this program for an input set (also known as fitness cases). The latter are not available in MCTS. We then extrapolate this idea to the fitness space. Thus, assuming we use a finite set of simulations, as normally adopted in MCTS, we can now define, without losing the generality, the semantics of a program in the simulations.

*Def. 2:* The semantics  $s(p)$  of a program  $p$  is the vector of values from each independent simulation  $sim$ ,

Thus, we have that the semantics of a program in MCTS is given by  $s(p) = [p(sim_1), p(sim_2), \dots, p(sim_l)]$ , where  $l = |I|$  is the number of independent simulations.

Based on Def. 2, we can define the *Sampling Semantics Distance* (SSD) between two programs  $(p, q)$ . That is, let  $P = \{p_1, p_2, \dots, p_N\}$  and  $Q = \{q_1, q_2, \dots, q_N\}$  be the sampling semantic of Program 1 ( $p_1$ ) and Program 2 ( $p_2$ ) on the same set of sample points, then the SSD between  $p_1$  and  $p_2$  is defined as  $SSD(p, q) = (|p_1 - q_1| + |p_2 - q_2| + \dots + |p_N - q_N|)/N$ , where  $S_p = \{p_1, \dots, p_N\}$  and  $S_q = \{q_1, \dots, q_N\}$  are the SS of programs  $p$  and  $q$  based on simulations.

We are now in position to use the well-known semantic similarity (SSi) proposed by the first author and colleagues [29]. This indicates whether the SSD between two programs lies between a lower bound  $\alpha$  and upper bound  $\beta$  or not. It determines if two programs are similar without being semantically identical. The SSi of two programs  $p$  and  $q$  on a domain is formally defined as

$$SSi(p, q) = (\alpha < SSD(p, q) < \beta) \quad (2)$$

where  $\alpha$  and  $\beta$  are the lower and upper bounds for semantic sensitivity, respectively. In our work, we set these 5 and 10, respectively.

#### E. Test Functions

We use five different functions, each of different degree of difficulty, going from unimodal functions, including multimodal functions to highly deceptive functions. These functions

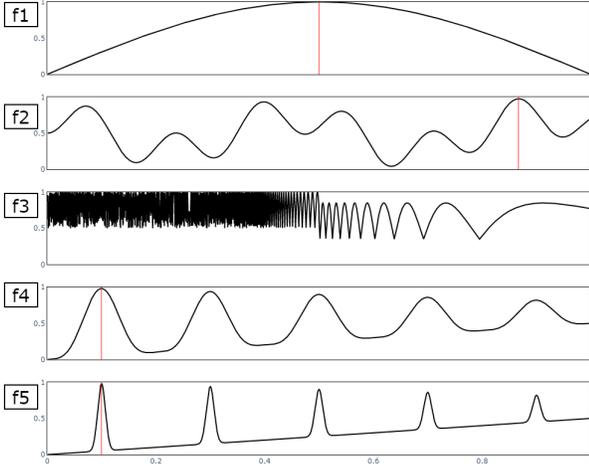


Fig. 1. Functions used for the function optimisation problem.

are shown in Figure 1, and for simplicity, we constrain the domain and range of the functions to be in the interval  $[0, 1]$ .

The first function, shown in Figure 1,  $f_1$ , depicts a unimodal function. The global optima is shown by a vertical red line. The function is defined by Eq. 3,

$$f_1(x) = \sin(\pi x) \quad (3)$$

The second function, shown in Figure 1,  $f_2$ , is a multimodal function, with a global optima situated in the right-hand side of the figure as denoted by the vertical red line. The function is defined by Eq. 4,

$$f_2(x) = 0.5 \sin(13x) \sin(27x) + 0.5 \quad (4)$$

The third function, inspired by [21], is shown in Figure 1,  $f_3$ . It shows a function with a high degree of ruggedness (left-hand side) containing multiple global optimum (not indicated as done with  $f_1$  and  $f_2$ ). In sharp contrast, the function also denotes smoothness in the right-hand side. The function is defined by Eq. 5,

$$f_3(x) = \begin{cases} 0.5 + 0.5 \left| \sin\left(\frac{1}{x^5}\right) \right| & \text{when } x < 0.5, \\ \frac{7}{20} + 0.5 \left| \sin\left(\frac{1}{x^5}\right) \right| & \text{when } x \geq 0.5. \end{cases} \quad (5)$$

The fourth function used in this work is shown in Figure 1,  $f_4$ . This is a deceptive function, where the global optimum is depicted by the red vertical line in the left-hand side of the figure. This function is defined by Eq. 6,

$$f_4(x) = 0.5x + (-0.7x + 1) \sin(5\pi x)^4 \quad (6)$$

The fifth and final function used in this work is shown in Figure 1,  $f_5$ . This is also a deceptive function, slightly harder compared to  $f_4$ . This function is defined by Eq. 7,

$$f_5(x) = 0.5x + (-0.7x + 1) \sin(5\pi x)^{80} \quad (7)$$

### III. AI CONTROLLERS

#### A. Monte Carlo Tree Search

The core idea of MCTS is based on its four functions: *Selection*, *Expansion*, *Rollout* and *Backpropagation*. The completion of these stages is known as a single simulation. When all simulations conclude, normally the node with the highest action value is chosen, which is the approach taken in this work. The MCTS is explained in detail in Section II.

#### B. Semantic-inspired Evolutionary Algorithm in MCTS

We now turn our attention to the proposed AI controller based on EAs to evolve online the mathematical expression to be used during the selection phase of the MCTS. To this end, we use  $(\mu, \lambda)$ -ES (see Section II). We first use the UCT formula as parent to later generate the offspring. We evolve a candidate solution in every turn that we need to make a decision. At each turn, a new evolved solution is built from scratch. Similar to [21], “in the simulation phase, actions are executed uniformly randomly until a terminal state is encountered, at which point some reward is received. Let  $f$  be the function and  $c$  be the midpoint of the state reached by the rollout. At iteration  $t$ , a binary reward  $r_t$ , drawn from a Bernoulli distribution  $r_t \sim \text{Bern}(f(c))$ , is generated”. This is how the fitness is assigned to each evolved potential solution. The value of these are used to update a *copy* of the MCTS statistical tree, from the selected node to the root including the nodes given in a branch. We perform 30 simulations to compute the fitness of the evolved expression. The fitness of our evolved individual is the average of these 30 simulations. We pick the offspring based on semantics to act as parent.

A potential major limitation in using a small population size, as adopted in this work, could be the lack of diversity leading to poor performance. To prevent this, we use semantics as inspiration to promote diversity (see Section II). We dubbed this method Semantic-inspired Evolution Algorithms in Monte Carlo Tree Search (SIEA-MCTS).

We first get the highest fitness,  $H_f$ , from the offspring. If there is more than one offspring with  $H_f$ , we compute the sampling semantic distance from each offspring with respect to Parent. We then proceed to compute the semantic similarity metric using thresholds. If there is more than one individual from the offspring population that falls within this threshold, defined by  $\alpha$  and  $\beta$ , then the individual closest to the  $\alpha$  value is picked. Otherwise, we select an individual randomly from the offspring population.

Our proposed method aims to evolve mathematical expressions that can replace UCT with the goal to get better or competitive results compared to UCT. Thus, ES is called during the selection step in MCTS. Once a node has been selected by our evolved expression, we proceed to compute the fitness of the evolved expression. We do so by performing roll outs as done in MCTS.

### IV. EXPERIMENTAL SETUP

#### A. Function and Terminal Sets

The terminal set is defined by  $T = \{Q(s, a), N(s), N(s, a), K\}$ , where  $N(s)$  is the number

TABLE I  
MCTS AGENTS PARAMETERS.

Parameter	Value
All MCTS agents	
C	$\frac{1}{2}, 1, \sqrt{2}, 2, 3$
Rollout playouts	1
Iterations	5000
SIEA-MCTS	
$(\mu, \lambda)$ -ES	$\mu = 1, \lambda = 4$
Generations	20
Type of Mutation	Subtree (90% internal node, 10% leaf)
Initialisation Method	Initial formula: UCB1
Maximum depth	8
Iterations	2,400 EA evaluations + 2,600 iterations

of visits to the node from the MCTS search tree,  $N(s, a)$  is the number of visits to a child node,  $Q(s, a)$  is the child's node action-value and  $K$  is the exploration-exploitation constant. When  $K$  is chosen to be mutated, it can take a random value from the following set  $r = \{0.5, 1, \sqrt{2}, 2, 3\}$ . The function set is defined by  $F = \{+, -, *, \div, \log, \sqrt{\cdot}\}$ , where the division operator is protected against division by zero and will return 1 for any divisor less than 0.001. Similarly, the natural log and square root operators are protected by taking the absolute values of input values. The values used for all our controllers are shown in Table I.

### B. Function Optimisation

The test functions used in this work, commonly known as function optimisation as described in [7], [21], is a problem where each state  $s$  represents a domain  $[a_s, b_s]$ , starting at  $[0, 1]$ . The available actions from state  $s$  are  $b$  evenly spaced partitions of the domain, sized  $\frac{a_s - b_s}{b}$  each, where  $b$  is the selected branching factor. The objective is to find the state where the global maxima of a given function  $f$  lies. A state is said to be terminal if its domain is smaller than the threshold  $t$ , in other words  $b_s - a_s < t$ . The threshold is set as  $t = 10^{-5}$  and the branching factor is set as  $b = 2$ .

The MCTS rollouts use a random uniform default policy. When a terminal state is reached,  $f$  is evaluated at the state's central point  $c_s = \frac{(a_s + b_s)}{2}$ . The reward  $r_s$  can either be 1 or 0 and is sampled from a Bernoulli distribution  $r_s \sim \text{Bern}(f(c_s))$ . Thus,  $0 \leq f(x) \leq 1 | x \in [0, 1]$  is ensured for every function  $f$ .

## V. DISCUSSION OF RESULTS

We are interested in knowing the effects of MCTS using different values of the UCT C constant and compared them with evolved functions that are used in the selection policy in lieu of the UCT formula. To do so, we keep track of how the search is carried out during the 5,000 iterations used by the MCTS and 2,600 iterations used by the EA (plus 2,400 evaluations). Thus, we will naturally observe a less number of nodes expanded in particular regions of a given formula when using the EA approach.

We divided these iterations equally by three. The first part is plotted with a light grey colour in Figures 2–6, the second part of this number of iterations is represented by a darker

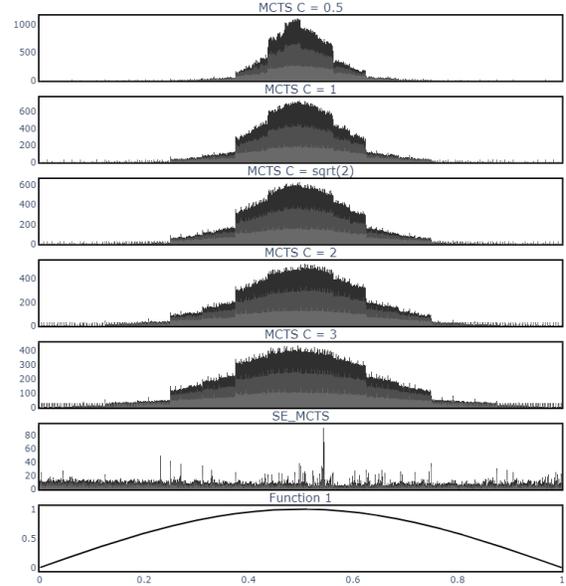


Fig. 2. Histogram of the location of the nodes for Function 1.

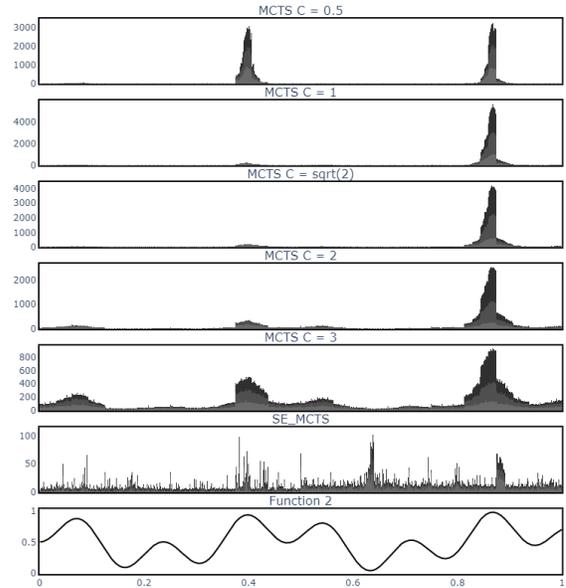


Fig. 3. Histogram of the location of the nodes for Function 2.

grey colour. Finally, the last part of is represented in black. In these figures, the  $x$ -axis correspond to the domain of the function and the  $y$ -axis corresponds to the number of nodes expanded in a particular region. The results are averaged from 30 independent runs for each AI controller. The histograms are generated by sorting the nodes of the tree into bins, according to the location of the center of their states. The goal is to illustrate how and when the nodes are expanded in different regions of the domain.

Let us start our analysis by analysing our first function (see Section II), re-plotted at the end of Figure 2 for convenience. This is an unimodal function and the easiest to be solved. The first five plots, from top to bottom, correspond to MCTS

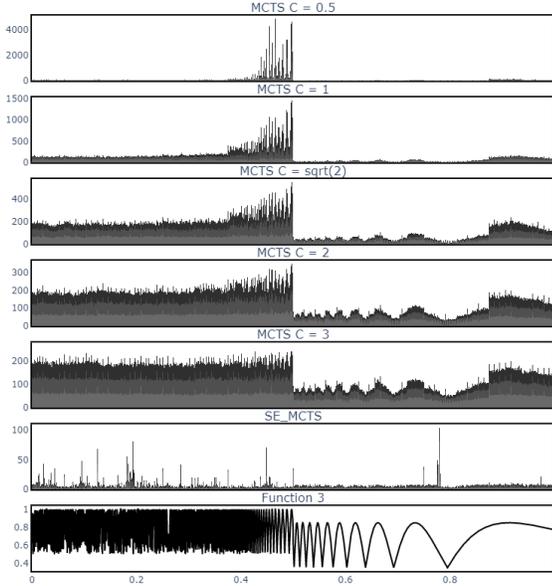


Fig. 4. Histogram of the location of the nodes for Function 3.

using  $C=\{0.5, 1, \sqrt{2}, 2, 3\}$ . As can be seen, as the  $C$  value increases, the MCTS explores larger regions. For instance, see the fifth plot, where  $C = 3$ , compared to the first plot, where  $C = 0.5$ . In the former case, there are close to 400 nodes around the global optima, compared to almost 1,000 nodes when using the latter. Let us now turn our attention to our EA method. We can see in plot sixth, from top to bottom, that this method has more variations compared to MCTS, regardless of the UCT  $C$  value used. This is expected given that at each node selection, we evolve an alternative UCT formula. Despite this, it is interesting to note that our EA method is able to expand a good number of nodes close to the global optima (center of the Function 1 as seen at the bottom of Figure 2).

Let us continue with our second function (see Section II), seen at the bottom of Figure 3. This multimodal function has also a global optima located in the right-hand side of the figure. When we focus our attention on how the MCTS behaves using this function (see first five plots from top to bottom in Figure 3), we can observe that as  $C$  is higher, the MCTS tends to expand nodes in the global optima as well as in local optima. This is depicted in the fifth plot, where there are three peaks. This situation changes as the UCT  $C$  value decreases, except when UCT  $C = 0.5$ , where two peaks are formed, one of them being where the global optima lies. Another point worth noting is the fact that the MCTS UCT  $C = \{1, \sqrt{2}\}$  exhibit a similar behaviour: both expand nodes around the global optima region as well as expanding a small number of nodes in a local optima region. When we turn our attention to the EA, we can see again a large variation in exploring/exploiting the search space. It has around seven peaks, one of them formed around the global optima region.

We now focus our attention on the third function used in this work (see Section II) and depicted at the bottom of Figure 4. This shows a high degree of ruggedness, left-hand side of the last plot, where multiple global optimum exist. In sharp

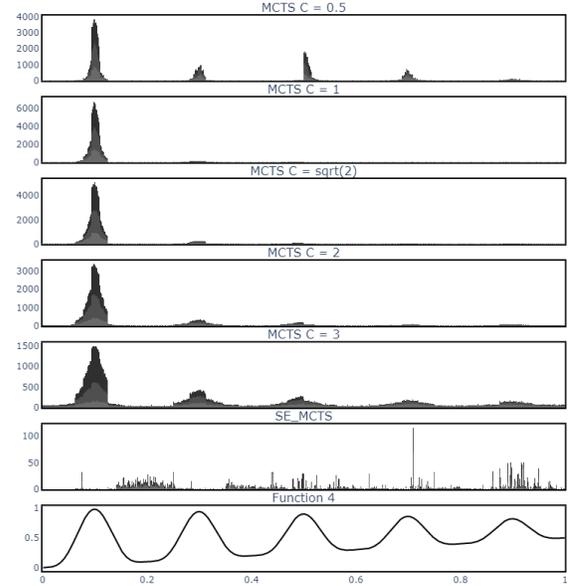


Fig. 5. Histogram of the location of the nodes for Function 4.

contrast, this function also denotes smoothness in the right-hand side. As can be seen in the first five plots, from top to bottom, of Figure 4, MCTS with a high UCT  $C$  value tends to explore more and so, the nodes tend to expand in ‘incorrect’ areas of the search space, as seen when  $C = \{2, 3\}$ . This is also visible when  $C = \{\sqrt{2}\}$ . The situation changes dramatically as UCT  $C$  decreases, for instance, see the first plot, where UCT  $C = 0.5$ . In this case there is a good number of nodes expanded in a narrow region where the global optimum lie. When we turn our attention to how the EA behaves, we can see an interesting trend: there is a good number of peaks, albeit small, nicely spread in the correct region where multiple global optimum are. This is in contrast to what is observed when using MCTS UCT  $C = 0.5$ , where some peaks are formed in just a fraction of the ‘correct’ part of the search space.

Let us focus our attention on the fourth function, defined in Section II and plotted at the bottom of Figure 5. This is a deceptive function, with a global optimum located in the left-hand side. Most of the MCTS perform well on this type of function. Although it is fair to say that MCTS UCT  $C = 1$  outperforms the rest of the UCT  $C$  values as it has the highest number of nodes expanded where the global optimum lies. When we proceed to analyse the behaviour of the EA approach, we can see that this exhibits some interesting behaviours. For example, it is able to spend some time in the region where the global optimum lies, although this is only a single line with a few number of nodes (around 35). The EA also spends some time in local optima as shown by the peaks formed as a result of the number of expanded nodes.

We finally proceed to focus our attention on the fifth function, defined in Section II, shown at the bottom of Figure 6. This is a deceptive function, harder compared to the fourth function, also deceptive, analysed previously. We can see that now the MCTS UCT  $C = 2$  performs better compared to the rest of the UCT  $C$  values used, since it is able to spend more

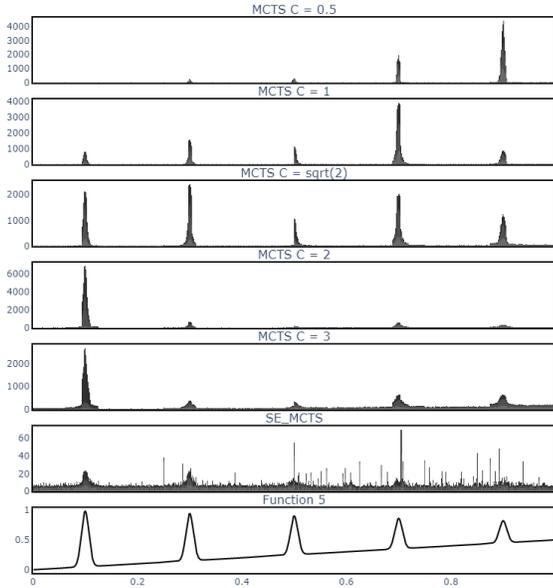


Fig. 6. Histogram of the location of the nodes for Function 5.

time in the global optimum located in the left-hand side of the function (bottom of Figure 6). This is in sharp contrast to what we observed in Function 4, as aforementioned, MCTS UCT  $C = 1$  performed better in this type of function. When we proceed to analyse the results of the EA, sixth plot from top to bottom of Figure 6, we can see that now the algorithm manages to spend more time in the correct area compared to the behaviour shown in Function 4 (a deceptive function too with smooth areas). Although it is fair to say that the EA also spends time in local optima, as shown by the peaks formed in other parts of the function.

From this analysis, we can see that MCTS UCT behaves very well in  $f_1$  and  $f_2$ . For the rest of the functions, it also performs well, although a correct UCT  $C$  value is needed as discussed before. For example, we have argued that for a deceptive function with some degree of smoothness ( $f_4$ ), the MCTS UCT = 1 behaves very well, but for a harder deceptive function ( $f_5$ ), the MCTS UCT  $C = 2$  is preferred. Perhaps this is one of the reasons why some researchers have carried out some research in evolving a selection policy that can be used in lieu of the MCTS UCT. When we observe the behaviour of the EA on the evolution of an alternative formula, we can see that this attains a poorer performance with MCTS UCT. However, it should be noted that in some cases, the use of the EA in MCTS can have a positive effect, such as in Functions 3 (high degree of ruggedness containing multiple global optima) and 5 (highly deceptive function).

## VI. CONCLUSIONS

Monte Carlo Tree Search (MCTS) is a sampling best-first method to search for optimal decisions. The success of MCTS depends heavily on how the MCTS statistical tree is built and the selection policy plays a fundamental role in this. A MCTS selection policy that works particularly well is the Upper Confidence Bounds for Trees, referred to as UCT. However,

some tuning is necessary for this to work well. Moreover, some sophisticated bounds have been proposed by the community to be used in lieu of the MCTS UCT.

As a result of this, some works e.g., [8], [5], [20] have proposed evolving the MCTS UCT in hope to get a better performing MCTS. Although all these works are inspiring, the entirety of these have focused their attention on a particular problem rather than a set of problems with certain features. This has limited generalising their findings and knowing under what circumstances the evolution of the MCTS UCT might be beneficial.

In this work, we have shown how the evolution of the MCTS UCT might be beneficial in multimodal scenarios as well as in deceptive ones. In contrast to this, the MCTS UCT performs incredibly well in unimodal scenarios and is competitive in the rest of the scenarios. Thus, we argue that it is important to know the features of the problem at hand when attempting to evolve the MCTS UCT to be used in lieu of the UCT commonly adopted in Monte Carlo Tree Search.

## ACKNOWLEDGMENTS

This work has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number SFI 18/CRT/6049.

## REFERENCES

- [1] A. M. Alhejali and S. M. Lucas. Using genetic programming to evolve heuristics for a monte carlo tree search ms pac-man agent. In *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, pages 1–8, 2013.
- [2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, 2002.
- [3] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996.
- [4] H. Baier and P. I. Cowling. Evolutionary mcts for multi-action adversarial games. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2018.
- [5] I. Bravi, A. Khalifa, C. Holmgård, and J. Togelius. Evolving game-specific ucb alternatives for general video game playing. In *EvoApplications, 2017*.
- [6] 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.
- [7] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvari. X-armed bandits. *Journal of Machine Learning Research*, 12:1655–1695, MAY 2011.
- [8] T. Cazenave. *Evolving monte-carlo tree search algorithms*. 2007.
- [9] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, 2003.
- [10] L. Fogel, A. Owens, and M. Walsh. *Artificial intelligence through simulated evolution*. Wiley, Chichester, WS, UK, 1966.
- [11] E. Galván and M. Schoenauer. Promoting semantic diversity in multi-objective genetic programming. In A. Auger and T. Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 1021–1029. ACM, 2019.
- [12] E. Galván-López, C. Harris, L. Trujillo, K. R. Vázquez, S. Clarke, and V. Cahill. Autonomous demand-side management system based on Monte Carlo tree search. In *IEEE International Energy Conference (EnergyCon)*, pages 1325 – 1332. IEEE Press, 2014.
- [13] E. Galván-López, R. Li, C. Patsakis, S. Clarke, and V. Cahill. Heuristic-based multi-agent monte carlo tree search. In *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*, pages 177–182. IEEE, 2014.

- [14] E. Galván-López, E. Mezura-Montes, O. A. ElHara, and M. Schoenauer. On the use of semantics in multi-objective genetic programming. In J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, editors, *Parallel Problem Solving from Nature - PPSN XIV - 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings*, volume 9921 of *Lecture Notes in Computer Science*, pages 353–363. Springer, 2016.
- [15] E. Galván and P. Mooney. Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Transactions on Artificial Intelligence*, 2(6):476–493, 2021.
- [16] E. Galván and G. Simpson. On the evolution of the mcts upper confidence bounds for trees by means of evolutionary algorithms in the game of carcassonne. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, 2021.
- [17] E. Galván, G. Simpson, and F. V. Ameneiro. Evolving the mcts upper confidence bounds for trees using a semantic-inspired evolutionary algorithm in the game of carcassonne. *IEEE Transactions on Games*, pages 1–10, 2022.
- [18] E. Galván, L. Trujillo, and F. Stapleton. Semantics in multi-objective genetic programming. *Applied Soft Computing*, 115:108143, 2022.
- [19] D. E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989.
- [20] C. Holmgård, M. C. Green, A. Liapis, and J. Togelius. Automated playtesting with procedural personas through MCTS with evolved heuristics. *IEEE Trans. Games*, 11(4):352–362, 2019.
- [21] S. James, G. Konidaris, and B. Rosman. An analysis of monte carlo tree search. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [22] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.
- [23] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [24] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
- [25] S. M. Lucas, S. Samothrakis, and D. Pérez. Fast evolutionary adaptation for monte carlo tree search. In A. I. Esparcia-Alcázar and A. M. Mora, editors, *Applications of Evolutionary Computation*, pages 349–360, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
- [26] I. Rechenberg. Evolution strategy: Nature’s way of optimization. In H. W. Bergmann, editor, *Optimization: Methods and Applications, Possibilities and Limitations*, pages 106–126, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.
- [27] M. P. D. Schadd, M. H. M. Winands, M. J. W. Tak, and J. W. H. M. Uiterwijk. Single-player monte-carlo tree search for samegame. *Knowl. Based Syst.*, 34:3–11, 2012.
- [28] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- [29] N. Q. Uy, N. X. Hoai, M. O’Neill, R. I. McKay, and E. G. López. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genet. Program. Evolvable Mach.*, 12(2):91–119, 2011.
- [30] F. Valdez-Ameneiro, E. Galván, and Á. F. K. Morales. Playing carcassonne with monte carlo tree search. In *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020, Canberra, Australia, December 1-4, 2020*, pages 2343–2350. IEEE, 2020.
- [31] L. Wang, Y. Zhao, Y. Jinnai, Y. Tian, and R. Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search, 2019.