Accelerating Empowerment Computation with UCT Tree Search

Christoph Salge
School of Computer Science
University of Hertfordshire
Hatfield, UK
c.salge@herts.ac.uk

Christian Guckelsberger
Computational Creativity Group
Goldsmiths, University of London
London, UK
c.guckelsberger@gold.ac.uk

Rodrigo Canaan
Tandon School of Engineering
New York University
Brooklyn, NY, USA
rodrigo.canaan@nyu.edu

Tobias Mahlmann
t.mahlmann@gmail.com

***Abstract*—Models of intrinsic motivation present an important means to produce sensible behaviour in the absence of extrinsic rewards. Applications in video games are varied, and range from intrinsically motivated general game-playing agents to non-player characters such as companions and enemies. The information-theoretic quantity of *Empowerment* is a particularly promising candidate motivation to produce believable, generic and robust behaviour. However, while it can be used in the absence of external reward functions that would need to be crafted and learned, empowerment is computationally expensive. In this paper, we propose a modified UCT tree search method to mitigate empowerment's computational complexity in discrete and deterministic scenarios. We demonstrate how to modify a *Monte-Carlo Search Tree with UCT* to realise empowerment maximisation, and discuss three additional modifications that facilitate better sampling. We evaluate the approach both quantitatively, by analysing how close our approach gets to the baseline of exhaustive empowerment computation with varying amounts of computational resources, and qualitatively, by analysing the resulting behaviour in a *Minecraft*-like scenario.**

***Index Terms*—empowerment, tree search, MCTS, UCT, Minecraft, intrinsic motivation**

I. Introduction

The empowerment formalism [1], [2] offers interesting game applications in terms of believable NPC behaviour [3], general game-play [4] and player experience modelling [5]. But the high computational complexity of empowerment is problematic for a wider application in games. In this paper we show how a UCT tree search formalism [6], [7] can be adapted to approximate empowerment maximisation in the discrete domain. But first, we motivate the application of empowerment to games in more detail.

A. Motivation

Empowerment is a measure of how much an agent can affect the world it itself perceives. Empowerment maximisation is considered an intrinsic motivation (IM) [8], and has been recently linked to competence and autonomy, two motivations which are frequently discussed in a games context [9]. As intrinsic motivation, i.e. as an essential motivation linked to agency itself, empowerment can generate behaviour even in the absence of externally defined goals. Behaviour then results from fulfilling a motivation that arises from the agent-world interaction. An illustrative, non-empowerment example for this is the work of Merrick and Maher [10], [11], where an agent's actions are selected based on learning progress [12] and curiosity [13]. Curiosity, or the desire to experience something new, can create behaviour without further reward. The broad concept of curiosity is also a good illustration of an *intrinsic* motivation, as it is hard to imagine agency without the least desire to experience novelty or learn something new. Empowerment, in contrast, is about having affordances, about self-efficacy and the ability to affect one's own world. Empowerment has also been linked to the idea of an organism's striving to preserve its precarious existence [14]. Applied to a Minecraft-like simulation this drive to self-preservation resulted in behaviour where the agent would restructure the world to keep itself alive [15], producing different behaviour patterns in reaction to changes in the environment. The same work also demonstrated how the embodiment of the agent was reflected in the structures built in the world. This apparent self-directed behaviour, arising from- and reacting to changes in the game world make the application of empowerment in games interesting.

Previously, empowerment has been applied to play Sokoban and PacMan [4]. In this work, Anthony *et al.* speak about the generality of empowerment by asserting that it provides a utility that: "1) derives only from the structure of the problem itself and not from an external reward; 2) identifies the desirability of states in a way that matches intuition; and 3) carries over between scenarios of apparently different character." This would make empowerment a good proxy for general game-play, and thus biasing the decision making of reward-optimising agents with empowerment might lead to better performance. Similar approaches have been used in the domain of robotics, where a robotic follower [16] and underwater vehicles [17] had their decision making biased or enhanced with empowerment maximisation. But while the empowerment formalism is generally applicable, i.e. can be computed based just on the structure of a given forward model, the utility it provides may not always agree with an externally defined reward. One common example here are games where empowerment has to be "traded away" to win a game - imagine a game like Starcraft where you first obtain resources and units, which increase your empowerment, but then you lose those units and resources in a material exchange to defeat

your opponent. Note though, that there is still an incentive to maximise empowerment, even as you are giving it up. It is also possible to define games with a (possibly contrived) win condition that conflicts with empowerment. In a lot of cases, though, games are designed to be aligned with intrinsic motivations, and progressing in a game usually goes along with increasing player empowerment. As you progress you usually get more abilities, better and more tools, have access to more actions, can explore more of the world and generally have more rather than less influence on the game world.

Creating believable non-player characters (NPCs) in a game is another possible application of empowerment in games. Empowerment maximisation can, without defined goals, produce behaviour related to self-preservation and maximisation of options. This can be used to give NPCs an appearance of self-determination. The general applicability of the formalism also allows NPCs to adapt to changing circumstances. This has been explored by Guckelsberger *et.al.* [3] for the design of general companion characters. In addition to having the companion NPC maximise their own empowerment, two additional empowerment drives were introduced. Maximising the player's empowerment motivates the companion to protect and help. It would, for example, shoot enemies that threaten the player. This multi-perspective approach has also been explored in relation to robots, where it could produce generic robot behaviour guidelines [18]. Very recently, it has been extended to design highly adaptive and robust adversary NPCs [19].

Finally, there is also the question as to which extent a player's empowerment in a game can be used as predictor for their experience. A preliminary study [5] identified causal efficacy as potential candidate experience that empowerment is closely related to, with mediate effects on "challenge, involvement, attention and engagement, learning and emotions". Having a measure that computes user experience without an actual player would be beneficial in rapid prototyping and when creating or adapting games automatically.

Yet, one downside of empowerment maximisation is its lack of scalability, due to the formalism's high computational complexity, especially when looking at longer time horizons. Approximations have been developed both for the *continuous* domain [20] and discrete but *noisy* models [4]. In this paper, we use UCT to accelerate the computation of the most empowered action in a *discrete and deterministic* model.

B. Overview

We first describe the actual empowerment formalism, and then focus on empowerment in discrete and deterministic models. We discuss the problems arising from sparse sampling, and introduce a modified UCT tree search algorithm to find the most empowered actions with less sampling. We complement this with three modifications – *novelty bias*, *aggregated empowerment* and *full branching* – to further enhance the sampling. We evaluate the optimisation scheme in a Minecraft-like world model which has also been used in previous work [15]. We briefly introduce the model, followed by a quantitative and a qualitative evaluation. We demonstrate how the UCT approach and the different modifications perform better with less samples than the current baseline of sparse random sampling.

II. Empowerment

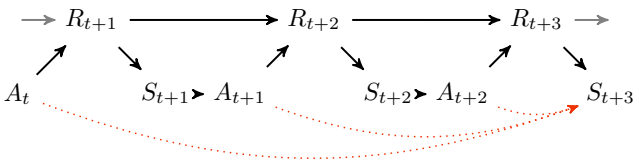

Figure 1. The perception-action-loop visualised as a Bayesian network. *S* is the sensor, *A* is the actuator, and *R* represents the rest of the system. The index t indicates the time at which the variable is considered. This model is a minimal model for a simple memoryless agent. The red arrows indicate the direction of the potential causal flow relevant for 3-step empowerment.

Empowerment [1], [15] is an information-theoretic formalism that captures how much an agent can affect the world it itself perceives. It is defined for all systems that can be modelled as an action-perception loop, as seen in Fig. 1. Where the random variables S, A and R model the sensors, actions and remaining state of the world, respectively. Empowerment for a given state $r \in R$ is formally defined as the channel capacity from an agent's actions at time t to its sensors at a later point in time. This channel goes through the environment R. A common generalisation is n-step empowerment, where all actions from a_t to a_{t+n-1} are considered as input to the channel, and the output is the sensor of the agent at $t+n$:

$$\mathfrak{E}(r_t) = \max_{p(a_{t\ldots t+n-1})} I(S_{t+n}; A_{t\ldots t+n-1}|r_t). \qquad (1)$$

The quantity captures how much information an agent can "inject" into its sensor S_{t+n} via the environment by intervening earlier in $A_{t\ldots t+n-1}$. It is equivalent to potential causal information flow as defined in [21]. An agent is usually highly empowered if it has a lot of different options that all lead to different, predictable outcomes, unaffected by noise. A highly empowered agent can reliably bring about many different sensor states. A more detailed discussion of the general concept and its information theoretic basis can be found in [1], [15].

Empowerment maximisation is the idea that an agent wants to be in a state that is highly empowered. Note, when computing the empowerment for a given state the channel capacity achieving distribution $p(a_{t\ldots t+n-1})$ might contain a lot of action sequences that lead to bad outcomes. But the action policy, i.e the way the agent picks it actions, is not determined by this distribution. Instead, a greedy empowerment maximisation strategy computes the empowerment for all possible successor states to the current states, and then chooses the action leading to the one with the most empowerment. It is empowerment maximisation that is considered an intrinsic motivation, and

in this paper, we focus on how to efficiently determine which actions will lead us to the most empowered state.

While empowerment can be defined for both a noisy and even a continuous channel, in this paper we focus on discrete and deterministic models. In the deterministic case, where each possible action sequence $a_t, ..., a_{t+n-1}$ leads to one specific state s_{t+n}, the channel capacity is the logarithm of the number all reachable states. So, to calculate the empowerment we have to determine the resulting sensor state for each possible n-step action sequence, and then count how many different states there are in total. This simplifies the computation significantly, and allows to compute n-step empowerment for larger time horizons n. We will refer to these reachable sensor states as "reachable states", dropping the word sensor for brevity. Despite this simplification, if we look at a model where each action step has, for example, a branching factor of 5, we still have to evaluate 5^n action sequences and the corresponding final states. This number quickly grows infeasibly. In previous work [15] this was addressed with sub-sampling, where a random subset of all possible action sequences was evaluated to compute 15-step empowerment, with a branching factor of 12. In this work the limitations of random sampling became evident. Sometimes, the empowerment-maximising agent would be in a situation where it could get to a part of the world where it would be able to reach a lot of different sensor states, but to get there it would have to perform very specific actions in the beginning of the action sequence - think of a bridge as an evocative example. Due to the random nature of the sampling, this bridge might only be crossed with a few of the sequences, and the evaluation would miss the considerable gain that going over the bridge yields for the agent's empowerment. In this work, we aim to use UCT tree search to both (i) bias the exploration towards those initial sequences, and to (ii) identify the best possible action more efficiently.

III. Empowerment with UCT Tree Search

In this section we outline how to use tree search with UCT (upper confidence bound applied to tree search, [6]) to accelerate the decision making based on deterministic empowerment. To find the best action, we need to determine which of the successor states of the current world state has the most different sensor states reachable with action sequences of length n. Note that while we compute the empowerment for sensor states derived from the world, the computations to determine the empowerment are done with a complete model of all world state transitions. In other words, our computation is not limited by the agent's perspective and has access to the full world model.

Our approach is inspired by Monte Carlo tree search (MCTS) with UCT [7], but there are substantial adaptations in the expansion, simulation and backpropagation steps. The basic idea of MCTS UCT, or any informed search for that matter, is to guide the use of resources, such as forward model calls, to the parts of the search space which yield the most information for picking the best action.

So, when we sample action sequences to determine reachable sensor states, we assume that sequences starting with actions that have led to new results are more likely to yield new results again. We then use the UCT formula [6] to bias our exploration towards those actions that have previously led to new states. Further analysis is still needed to determine if the mathematical properties of the bandit problem, which UCT is derived from, hold for empowerment computation. Here we look at simulated results only.

In the next section, we describe the algorithm in detail, and motivate three modifications. Both are further illustrated with pseudocode in Alg. 1. Keep in mind, the goals is to determine which successor state of the current world state has the most reachable sensor states, i.e. is the most empowered.

A. UCT tree search

In our algorithm nodes are associated with world states. We start by creating a root node that is associated with the world in its current state. This node has a depth of zero.

1) Expansion: The algorithm starts at the root, and checks if there are unexpanded, i.e. unvisited children. As long as there are unvisited children we select randomly one of the actions that would lead to an unexpanded child. We create the child node with a depth value one higher than its parent node. We then repeat the expansion step, i.e. expanding another unvisited child, until we reach a node that has a depth equal to the empowerment horizon n plus one. This is because the first step just creates the successor nodes that we are evaluating for their empowerment, and the successive n steps realise the n-step empowerment approximation. Note that we immediately expand the tree down to n-steps, and there is no roll-out phase.

2) Backpropagation: Once we reach a node at the horizon, we obtain the agent's sensor state and store it in the set of reachable states in the node. We then also add this reachable state to the node's parent node. The parent checks if it already has this state in its reachability set, and if not, adds it. It then also adds it to its own parent, recursively. After the backpropagation finishes, the tree should be in a state where each node has a reachability set that contains all sensor states that can be reached from it with the already expanded action sequences. After this step the algorithm starts over at the root of the tree, if there is time left.

3) Selection: As the tree fills up, the algorithm will encounter nodes where all children have at least been visited once, and it then has to decide which node to expand again. At this point we sort the children c with the modified UCT formula and pick the child c with the highest value:

$$uct(c) = \frac{c.states.size()}{c.visits} + 0.01 \cdot \sqrt{\frac{\log\left(root.visits\right)}{c.visits}} \quad (2)$$

The function $states.size()$ gives us the size of the reachability set, and $visits()$ tells us how often the root node and the child node have been visited. The size of the reachability set divided by the visits to the child gives us a value between 0.0 and 1.0, the ratio of how many new states each visit to this

Algorithm 1 Overview of the agent's decision making algorithm. Methods that take place in other objects, such as applying actions to the world or selecting a random number are omitted for brevity. HORIZON is $n+1$ for n-step empowerment, as we can determine the initial successor states in the same tree. The colours indicate code changes for the modification. For basic UCT empowerment read just the code in black. For *aggregated empowerment*, add the red line at 18. For *novelty bias* add the two blue code snippets in 29 and 36. For the *full branching* modification add the code in green, and set DEPTH to a non-zero value.

```
1:  procedure BEST_ACTION(World w)
2:      Node root ← new Node
3:      while time left do
4:          depth ← 0
5:          Node t ← root
6:          World test ←copy(w)
7:          while depth < HORIZON - DEPTH do
8:              t.visits++
9:              depth++
10:             if t has unexpanded children then
11:                 Action a ← RANDOM_ACTION(t, test)
12:                 test.applyAction(a)
13:                 Node child ← new Node
14:                 child.parent ← t
15:                 child.action ← a
16:                 t.children.add(child)
17:                 state ← test.s()          ▷ Agent sensor state
18:                 ADD_STATE(state, child)
19:                 t ← child
20:             else                          ▷ use UCT selection
21:                 t ← BEST_CHILD(t, root)
22:                 test.applyAction(t.action)
23:         BRANCH(t, test, DEPTH, root)
24:     return root.children[max(states)].action
25:
26: procedure ADD_STATE(state, Node n)
27:     if state ∉ n.states then
28:         n.states.add(state)
29:         if state ∉ parent.states then n.unique++
30:         if ∃ n.parent then ADD_STATE(state, n.parent)
31:
32: procedure BEST_CHILD(Node t, Node root)
33:     best ← null
34:     fitness ← 0
35:     for c ∈ t.children do
36:         f = (|c.states| + c.unique)/c.visits + 0.01 · sqrt(log root.visits / c.visits)
37:         if f > fitness then fitness ← f; best ← c
38:     return best
39:
40: procedure BRANCH(Node t, World w, d, Node root)
41:     if d = 0 then
42:         state ← test.s()                  ▷ Agent sensor state
43:         ADD_STATE(state, t, root)
44:     else
45:         Action[] a ← w.getPossibleActions()
46:         for action ∈ a do
47:             World test ← copy(w)
48:             test.applyAction(action)
49:             Node child ← new Node
50:             child.parent ← t
51:             child.action ← action
52:             t.children.add(child)
53:             BRANCH(child, test, d - 1, root)
54:
55: procedure RANDOM_ACTION(Node t, World w)
56:     Action[] a ← w.getPossibleActions()
57:     for actions ∈ t.children do a.remove(action)
58:     return random(a)
```

particular child found. The best value here is 1.0, meaning every visit leads to one new sensor state. The term in the square root guides selection towards under-explored states; it grows larger as more action sequences are sampled that do not go through c. The factor of 0.01 was chosen to have a good balance where the distribution of visits among children was neither approximately uniform (as it would be for a larger value), nor heavily skewed towards the first best solution (as it would be for a smaller value).

4) Action Selection: At some point the algorithm runs out of time (or some other computational limiter) and it needs to decide what action to take. It looks at the root and picks the action leading to the child with the biggest reachability set. This action should lead to a state with the highest n-step empowerment, meaning that from this state, the agent can reach the maximum number of different sensor states within n steps. Note that the algorithm does not select the state with the highest ratio of new states vs. visits, which was used for the UCT based selection.

Rationale: Compared with randomly sampling action sequences for each successor state, this algorithm should perform better at finding the state with the highest empowerment, as it spends less time on investigating the worst alternatives. The UCT function in selection should bias computational resources towards the most promising candidates. While the utility of a node for action selection is the size of the reachability set, we chose to divide this value by the number of visits for the selection phase. We thus keep the value between 0.0 and 1.0 and the nodes remain comparable. If this was not the case, nodes that had been explored more would be preferred as the number of found reachable states heavily depends on the number of visits.

Before we evaluate this approach, referred to as UCT from now on, we introduce three modifications to the algorithm.

B. UCT with Novelty Bias

For this modification we check every time a child node adds a reachable state to its parents reachability set, if this state is new to the set. If this is the case, no other child of that parent has an expanded action sequence leading to that sensor state

yet. In case of a novel addition, the child increases its novelty counter by one (line 29 in Alg. 1).

The novelty counter is added to the size of the reachability set in the UCT function. This means that the performance of the nodes is now not only defined by how many different states it found, but also by how many of those states were novel contribution to the reachability set of its parent node; an idea somewhat similar to novelty pruning[22].

Rationale: This modification is aimed at biasing exploration towards sub-sequences that add novel states. Consider an agent in front of a doorway. One action leads through it, while the others keep the agent moving around in the room. The sequences that stay in the room end up overlapping, and while they might each lead to a sizeable amount of states, they all lead to the same states. In contrast, sequences that go through the door can add new reachable states to the parent nodes, and should therefore be preferred.

C. Aggregated Empowerment

For aggregated empowerment, which we will refer to as UCTa, we not only consider the sensor states reached at tree depth n, but we also consider all reachable states along the way. This is achieved by simply adding the sensor state of the current world state to every newly expanded node and propagating it upward (line 18 in Alg.1).

Rationale: Aggregated empowerment corresponds to a somewhat different quantity, and the implications of this measure go beyond the scope of this paper. Using this value to approximate regular empowerment however still has the advantages that it allows to differentiate between individual action sequences. Normally, each sequence reaches exactly one state. But with aggregated empowerment, sequences that go through different sensor sequences along the way are considered better. This difference can indicate that the agent is still operational and able to affect the world.

More conceptually, it also allows to differentiate between different sequences, even if they ultimately end in the same state, e.g. death. In this case, the agents prefers to live a less boring life. In a way, this introduces a form of count-based novelty into the empowerment calculation.

D. UCT with Full Branching

For n-step empowerment with k-step *full branching*, the algorithm only expands the tree to a depth of $n - k$, i.e. it stops k steps before it reaches the time horizon. It then fully expands the tree from that node for the remaining k steps with depth-first search, and eventually propagates all found sensor states upwards. All full branching examples in this paper use 1-step full branching. This modification is implemented by the extension of the **BRANCH** function in Alg.1, highlighted in green.

Rationale: Full branching also aims to differentiate between sequences. It basically computes 1-step deterministic empowerment for the semi-leaf node. This local empowerment ideally gives us an idea of how empowered close-by states are, and thus should guide our exploration towards- or away from action sequences starting with the same actions.

IV. Evaluation

A. Simulation Model

To evaluate the different UCT algorithms we implemented a three-dimensional block world, similar to [15]. The world is a three dimensional grid, and each grid cell is empty or contains exactly one block such as earth, the agent and lava. The agent can try to move in the four cardinal directions (*north, east, west, south*). The move will be successful, if a) the target location is empty, or if b) the target location is filled, but the one above is empty. In the second case, the agent will 'climb' to the location above the target block. In all other cases the move fails (is blocked) and the agent remains in its current position. The agent can also *act* in all six directions (*above, below, north, south, east, west*). This action is context-sensitive on the agent's inventory, which can hold exactly one block. If the inventory is empty, the agent will try to take the block from the target location into its inventory, if there is any. If the inventory contains a block, the agent will try to place it. This will succeed if the target location is empty. If an action fails, the world remains unchanged. There are two additional actions, *waiting* and *destroying the current block* in the inventory. Overall, this gives the agent 12 actions in each time step.

Between actions the world simulates liquid flow and gravity. Agents and lava are affected by gravity, i.e. they fall until they rest above a filled block. Earth blocks are not affected by gravity. Lava spreads to neighbouring tiles every 4 time steps, if they are empty. Lava is an environmental hazard, and the agent dies if it is next to a lava block. 'Death' in this case means that the agent's actions have no effect any more on the world. The sensors considered for the empowerment computation capture the agent's x, y, z position.

B. Method

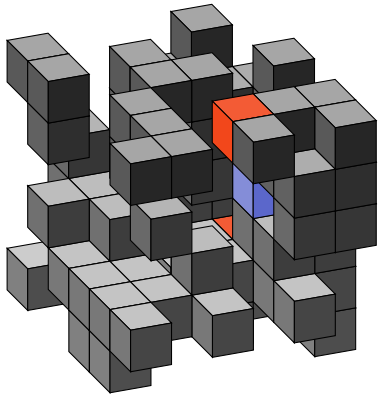

Fig. 2: Typical randomly generated world used for the quantitative evaluation. Two red lava blocks are visible among the grey earth blocks. The agent is colored blue.

For the quantitative evaluation we created 1000 different random worlds of size 7x7x7. Each block has a 40% chance to be an earth block, a 2% chance to be a lava block, and remains empty otherwise. Fig. 2 shows an example world.

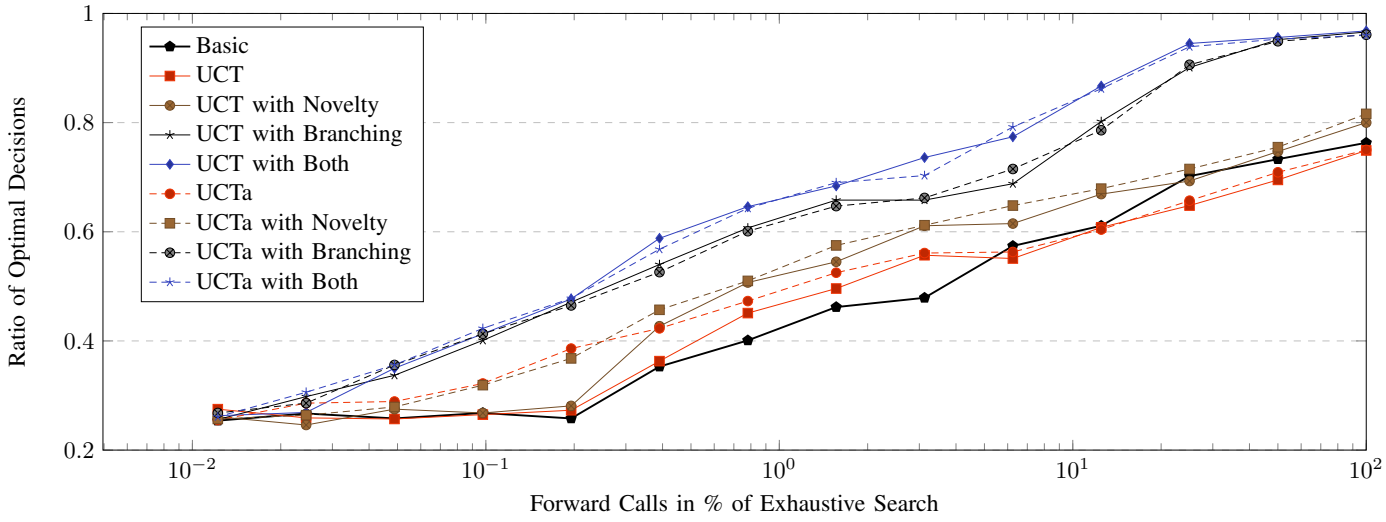

Fig. 3: Ratio of selecting the optimal action for different budgets of forward model calls. The evaluation is based on 1000 different random worlds, computing 4-step action sequences.

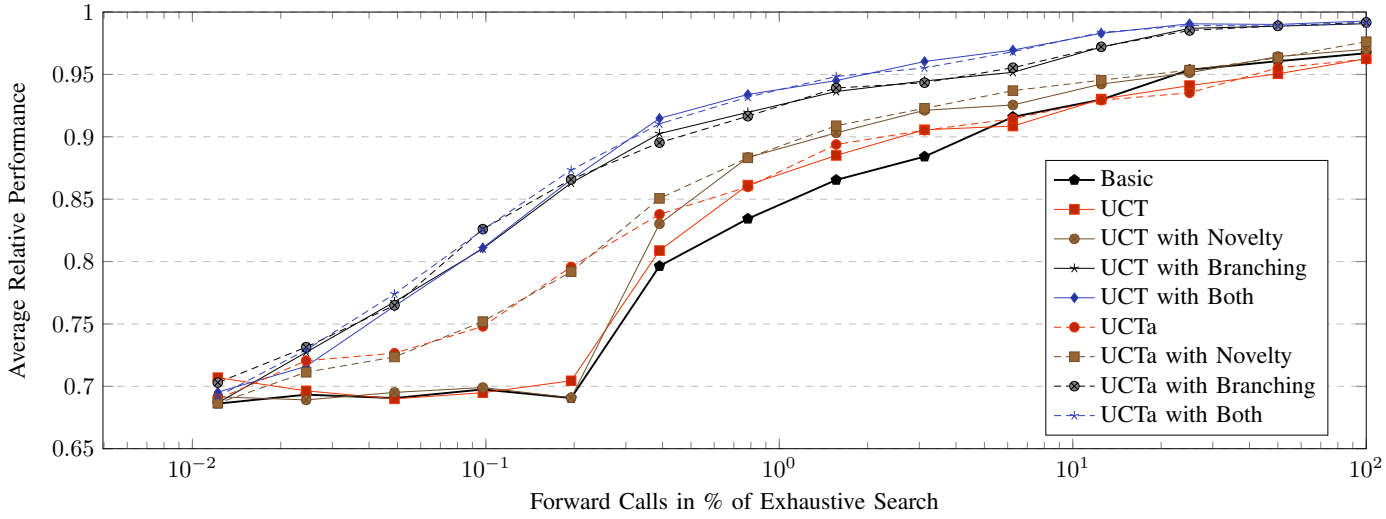

Fig. 4: Average relative performance (in reachable states) for the picked actions compared to the optimal action, for different budgets of forward model calls. The evaluation is based on 1000 different random worlds, computing 4-step action sequences.

The agent (blue) is placed in a random position, replacing the block in that position if necessary.

For each world we computed 4-step empowerment with an exhaustive depth-first search to obtain a baseline comparison. We recorded the number of reachable states, which determine the empowerment, for each immediate successor state and the action leading to that state from the root. This gives us an empowerment value for each action. The UCT algorithm can ultimately be applied to longer action sequences, but we chose a horizon of 4 for the quantitative evaluation, as it allows us to compare the approximations to an exact baseline.

We evaluated 9 different algorithms. The *basic* agent realises just random sampling, as describe in [15]. We evaluated the described *UCT* algorithm with each of the three modifications (*novelty bias*, *aggregated empowerment*, *full branching* for 1 step) turned on or off. Additionally, we evaluated the *UCTa* algorithm based on aggregated empowerment for each modification.

For each of the 1000 different worlds, each of the 9 algorithms tried to find the most empowered action. Our goal was to see how the algorithm's performance would degrade if they were given less computational resources. To have a hardware-independent comparison, we limited each algorithm to a certain number of forward model calls, which are used every time a world is advanced by applying an action. This decision was based on a preliminary analysis, showing that forward model calls were (unsurprisingly) the main contributor to computational load. We recorded the number of forward model calls used by the depth-first algorithm from the exhaustive baseline computation. This number, 22621, was considered as

100% of needed calls. We then evaluated all 9 algorithms on all 1000 worlds by giving them only $1/2$, $1/4$, $1/8$, etc, of the forward calls required for the baseline.

The performance of returned actions was evaluated in two ways. We checked, in comparison to the baseline computation, if the action selected by an action is optimal, i.e. if it has as many reachable states as the best answer. The average performance in this case is the ratio of how many of the chosen actions where optimal. We also implemented a second performance measure, where we compared the empowerment of the given action to the empowerment of the best action. If for example the best action would lead to a state with 20 reachable states, and the given answer would lead to a state with 10 reachable states, the performance would be 0.5. The performance is then averaged over the 1000 tested worlds.

C. Results

Figs. 3 and 4 show the results of the evaluation. As expected, both performance measures get worse if the agents get fewer forward calls to determine the best action. The important result here is that the UCT algorithms, especially those with modifications, outperform the basic agent from [15] for less forward calls. If we look at the graph in more detail, we see that for the least number of samples, all agents perform at about 0.7 in Fig. 4 and at about 0.25 in Fig. 3. This is basically the performance of a random agent that occasionally finds the best action by chance, or picks an action that has decent empowerment. Keep in mind that in some worlds several actions are optimal, or close to optimal.

The remaining graph can be split into three parts. For the first 5 sample points, i.e. the part with the least forward calls, we see that the basic agent —●—, as well as UCT —■— and UCT with novelty —●— all perform at the random level. At this point so few forward calls are made that each child of the root is only expanded once, i.e. has one full actions sequence going through it evaluated. This does not allow yet for an informed choice, as all sequences in this case lead to exactly one sensor state, making all choices equivalent. The algorithms with aggregated empowerment (-●-,-■-,-⊗-,-✶-) perform better here, as even a single sequence becomes informative. Each sequence might be visiting more or fewer different states, giving a better picture of how much an agent that picked a specific first action can affect the world afterwards. If, for example, the first action would lead the agent to touch lava and die, or fall in a hole, then all successive states of that sequence would be identical. The four algorithms with full 1-step branching (—✶—,—♦—,-⊗-,-✶-) perform best in the segment, as they create the most information for a single sequence. By evaluating how many states could be reached from the second to last step, the algorithms can differentiate between different starting actions. These approaches cost slightly more, as they use 11 additional forward calls in the end, but then saves forward calls by not having to go down the same full sequence several times.

In the middle segment of the graph we see an increase in performance for all algorithms. Noteworthy here is that for every configuration the variant with novelty outperforms the respective variant without novelty bias. As we now have enough samples so that children are fully expanded, the UCT selection comes into play. The bias towards those states that brought novel contributions seems to guide us towards better states. Aggregated Empowerment seems to have little effect on performance in this segment.

Towards the end, where the algorithms get nearly enough calls to exhaustively sample the space we can see that the marginal difference in performance gets slimmer as the performances increase overall. The algorithms basically split into two groups, those with full 1-step branching (—✶—,—♦—,-⊗-,-✶-) clearly outperforming those without. We should also note that the basic agent performing random sampling gets more competitive once we get to nearly 100% of the needed samples.

D. Bridge Example

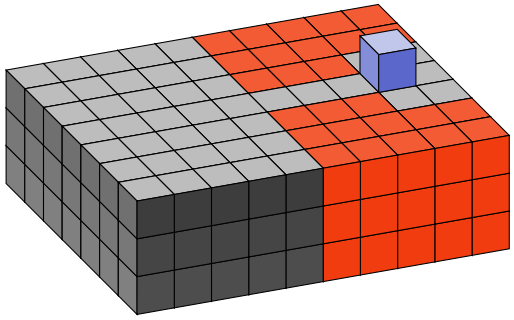

Fig. 5: Bridge example world. The agent (blue) should cross over the narrow bridge and avoid the lava (red).

The qualitative effect of the different algorithms becomes more evident if we are looking at a concrete example. In Fig. 5 we see an agent on a small platform surrounded by lava. The agent should cross over the narrow bridge to reach the much larger area. This would massively increase its empowerment. Looking at 10-step empowerment, the agent has a long enough time horizon to figure this out. But it is infeasible to evaluate all 61,917,364,224 10-step action sequences, so we do not have a ground truth to evaluate against. This is the kind of scenario the algorithm in this paper was developed for.

We evaluated this example with 10000 forward calls and less, for all previously described algorithmic variations. We ran the world seen in Fig. 5 100 times, and we checked how often the agent started moving onto the bridge leading to more empowerment. From our conceptual understanding of discrete empowerment we know that this is the most empowered option. The graph in Fig. 6 shows the results of the evaluation. The basic —●— and the UCT agent —■— struggle to find the optimal action, as fewer than 5% actually move towards the bridge with the first action. The bridge defines a bottleneck that first has to be crossed with an initial, very specific 4-step sequence, and biasing the exploration of the graph towards this one sequence does not happen in this case. The addition of novelty bias seems to add relatively little, but full branching

and aggregated empowerment pick the optimal action far more often. The algorithm that combines all modifications performs best, finding the optimal solution 52% of the time compared to 2% for the basic agent from [15]. As a side note, this is also an example of how directed behaviour arises in a world without the kind of utility function typical to most games. The agent prefers to be on the bigger platform, as it can move around more and dig down to obtain blocks to climb up.

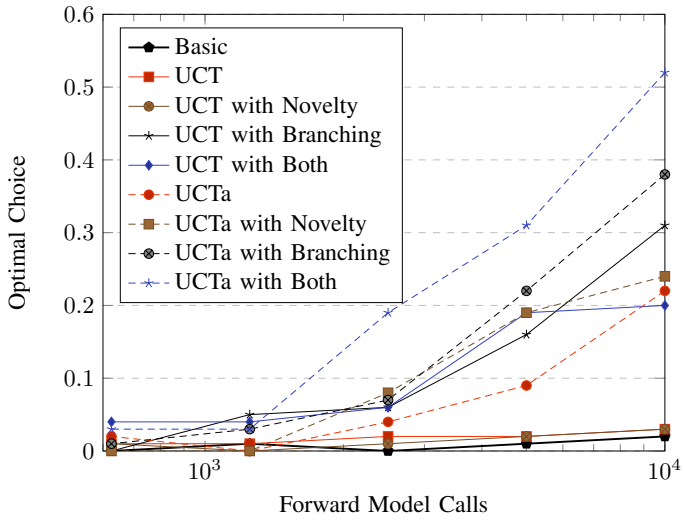

Fig. 6: Evaluation of the bridge example in Fig. 5 for 10-step sequences. The optimal choice is going towards the bridge.

V. Conclusion

The results indicate that all three modifications introduced in this paper improve the performance of empowerment maximising agents that have a limited amount of computational resources. While the approach is intended to be used for longer, i.e. above 10-steps, action sequences, we evaluated 4-step sequences. Due to computational intractability, we do not have a baseline comparison for longer sequences. The bridge example is promising though, as it showed that the modified algorithms can deal with bottlenecks in longer sequences, something the basic algorithm from previous work [15] struggled with considerably.

A. Future Work

This technical improvement opens up possible applications of empowerment in the discrete and deterministic domain. For example, this would make it more feasible to apply an empowerment-biased agent to games such as those found in general game-playing competitions [23], [24]. The other possible extension of this work is to extend it to non-deterministic but discrete models. While there are faster approximations for this domain [4], it might prove useful to apply the extensive catalogue of MCTS enhancements to this problem.

Acknowledgement

CS is funded by the EU Horizon 2020 programme under the Marie Sklodowska-Curie grant 705643. CG is funded by EPSRC grant [EP/L015846/1] (IGGI). RC gratefully acknowledges the financial support from Honda Research Institute Europe (HRI-EU). We thank the anonymous reviewers for helpful feedback.

References

[1] A. S. Klyubin, D. Polani, and C. L. Nehaniv, "Keep Your Options Open: An Information-Based Driving Principle for Sensorimotor Systems," *PloS one*, vol. 3, no. 12, pp. 1–14, 2008.

[2] C. Salge, C. Glackin, and D. Polani, "Empowerment – an Introduction," *Guided Self-Organization: Inception*, pp. 67–114, 2014.

[3] C. Guckelsberger, C. Salge, and S. Colton, "Intrinsically Motivated General Companion NPCs via Coupled Empowerment Maximisation," in *Proc. Conf. CIG*. IEEE, 2016, pp. 1–8.

[4] T. Anthony, D. Polani, and C. L. Nehaniv, "General Self-Motivation and Strategy Identification: Case Studies based on Sokoban and Pac-Man," *Trans. CIAIG*, vol. 6, no. 1, pp. 1–17, 2014.

[5] C. Guckelsberger, C. Salge, J. Gow, and P. Cairns, "Predicting Player Experience Without the Player.: An Exploratory Study," in *Proc. Symp. Computer-Human Interaction in Play*. ACM, 2017, pp. 305–315.

[6] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Machine Learning: ECML 2006*. Springer, 2006, pp. 282–293.

[7] 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," *Trans. CIAIG*, vol. 4, no. 1, pp. 1–43, 2012.

[8] P.-Y. Oudeyer and F. Kaplan, "How Can We Define Intrinsic Motivation?" in *Proc. 8th Conf. Epigenetic Robotics*, 2008, pp. 93–101.

[9] S. Roohi, J. Takatalo, C. Guckelsberger, and P. Hämäläinen, "Review of Intrinsic Motivation in Simulation-based Game Testing," in *Proc. 36th ACM Conf. Human Factors in Computing Systems*, 2018.

[10] K. Merrick, "Modeling Motivation for Adaptive Nonplayer Characters in Dynamic Computer Game Worlds," *CiE*, vol. 5, no. 4, p. 1, 2008.

[11] K. E. Merrick and M. L. Maher, *Motivated Reinforcement Learning. Curious Characters for Multiuser Games*. Springer, 2009.

[12] P. Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic Motivation Systems for Autonomous Mental Development," *IEEE Trans. Evolutionary Computation*, vol. 11, no. 2, pp. 265–286, April 2007.

[13] J. Schmidhuber, "Developmental Robotics, Optimal Artificial Curiosity, Creativity, Music, and the Fine Arts," *Connection Science*, vol. 18, no. 2, pp. 173–187, 2006.

[14] C. Guckelsberger and C. Salge, "Does Empowerment Maximisation Allow for Enactive Artificial Agents?" in *Proc. Conf. ALIFE*, 2016.

[15] C. Salge, C. Glackin, and D. Polani, "Changing the Environment Based on Empowerment as Intrinsic Motivation," *Entropy*, vol. 16, no. 5, pp. 2789–2819, 2014.

[16] A. Leu, D. Ristić-Durrant, S. Slavnić, C. Glackin, C. Salge, D. Polani, A. Badii, A. Khan, and R. Raval, "CORBYS Cognitive Control Architecture for Robotic Follower," in *Proc. Symp. System Integration*. IEEE/SICE, Dec 2013, pp. 394–399.

[17] P. Abreu, G. Antonelli, F. Arrichiello, A. Caffaz, A. Caiti, G. Casalino, N. C. Volpi, I. B. De Jong, D. De Palma, H. Duarte *et al.*, "Widely Scalable Mobile Underwater Sonar Technology: An Overview of the H2020 WiMUST project," *Marine Technology Society Journal*, vol. 50, no. 4, pp. 42–53, 2016.

[18] C. Salge and D. Polani, "Empowerment As Replacement for the Three Laws of Robotics," *Frontiers in Robotics and AI*, vol. 4, p. 25, 2017.

[19] C. Guckelsberger, C. Salge, and J. Togelius, "New And Surprising Ways to be Mean: Adversarial NPCs with Coupled Empowerment Minimisation," in *Proc. Conf. CIG*. IEEE, 2018.

[20] C. Salge, C. Glackin, and D. Polani, "Approximation of Empowerment in the Continuous Domain," *Advances in Complex Sys.*, vol. 16, 2012.

[21] N. Ay and D. Polani, "Information Flows in Causal Networks," *Advances in complex systems*, vol. 11, no. 01, pp. 17–41, 2008.

[22] D. J. Soemers, C. F. Sironi, T. Schuster, and M. H. Winands, "Enhancements for real-time monte-carlo tree search in general video game playing," in *Proc. Conf. CIG*. IEEE, 2016, pp. 1–8.

[23] M. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI Magazine*, vol. 26, no. 2, p. 62, 2005.

[24] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, "General Video Game AI: Competition, Challenges and Opportunities," in *Proc. Conf. AAAI*, 2016, pp. 4335–4337.