UCT for PCG

Cameron Browne
Imperial College London
South Kensington, UK
camb@doc.ic.ac.uk

***Abstract*—This paper describes initial experiments in the use of UCT-based algorithms for procedural content generation in creative game-like domains. UCT search offers potential benefits for this task, as its systematic method of node expansion constitutes an inherent form of exhaustive local search. A new variant called *upper confidence bounds for graphs* (UCG) is described, suitable for bitstring domains with reversible operations, such as those to which genetic algorithms are typically applied. We compare the performance of UCT-based methods with known search methods for two test domains, with encouraging results.**

I. Introduction

Procedural content generation (PCG) involves the automated generation of desirable content for games [1]. This is a form of *creative search*, in which the aim is to find a range of interesting examples of the domain being modelled. We contrast this with *optimising search* – the standard form of search used in AI – in which the aim is typically to converge to a single optimal solution.

In the field of *computational creativity*, Ritchie [2] describes three properties that the artefacts generated by a system should possess for that system to be considered "creative":

1) *Typicality*: Artefacts should be typical of the domain being modelled.
2) *Quality*: Artefacts should be valuable within the context of this domain.
3) *Novelty*: Artefacts should be novel compared to previous output.

For example, a puzzle designer may wish to generate a booklet of challenges, or a level designer may wish to generate an effectively endless variety of levels for an online game. It is important that the generated artefacts be: 1) typical of the game being modelled, 2) engaging for the player, and 3) novel enough to maintain the player's interest. Rather than *converging* to a single optimal solution, the designer wants the search to *diverge* to a number of interesting new solutions.

This paper compares the operation of some algorithms typically used for creative search on a graphic design domain and a puzzle generation domain, and explores the use of the *upper confidence bounds for trees* (UCT) approach for this purpose. UCT is an optimising search that has had considerable success with optimal move planning for games [3], but we demonstrate how some of its inherent properties can also be exploited for effective creative search. Section II compares existing search methods, Section III describes a new search method called UCG, and Section IV compares the performance of the various searches for two creative tasks.

II. Search Methods

In this paper, we focus on domains modelled as *bitstrings* with length not exceeding B=64 bits. This section describes the main search methods that we will compare for creative search in such domains.

A. Evolutionary Approaches

Genetic algorithms (GA) are the standard evolutionary search method for bitstring domains. The basic operation is *mutation* through bit flipping, which is *reversible*. i.e. performing the same action twice in succession on a state s will return it to its original state $s \to s' \to s$:

$$\mathsf{0000} \iff \mathsf{0001}$$

Any state in the bitstring space can be reached from any other through a combination of such operations, in any order. Bitstring domains therefore constitute non-planar graphs, with each vertex describing a state s and each edge a transition a between adjacent states. For example, Figure 1 shows Hasse diagrams for bitstring domains of sizes $B = 1..4$.

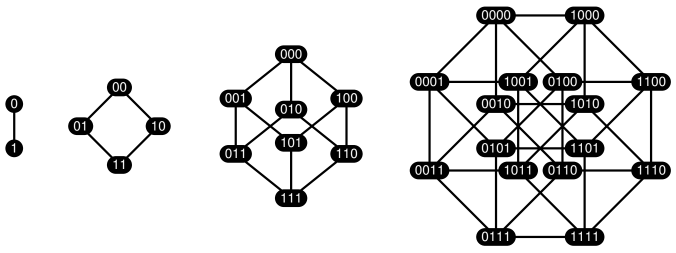

Fig. 1. Hasse transition diagrams for bitstrings of sizes $B = 1..4$.

Maintaining diversity in populations can be an issue [4].

B. Monte Carlo Approaches

Monte Carlo (MC) simulation involves sampling solutions from the search space uniformly at random, which typically produces a diverse range of solutions with no guarantees regarding fitness. This is the simplest form of search with the least computational overhead, and is used as a baseline for comparison.

Monte Carlo tree search (MCTS) is a family of algorithms in which a search tree is incrementally built in memory based on repeated MC simulations [3]. The main steps are shown in Figure 2. Each node in the tree represents a *state* and each edge represents an *action* leading to a subsequent state.

Firstly, a *tree policy* is applied to descend from the tree's root to its *most urgent* node (Figure 2, left), which is typically

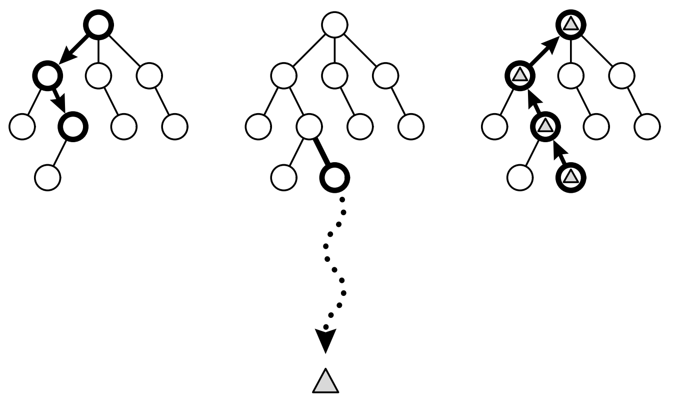
Fig. 2. Basic steps of Monte Carlo tree search.

the first node encountered with unexplored children. The tree is expanded to an unexplored child of the most urgent node, whose value is estimated using a *default policy* such as MC simulation (middle). This value is then backpropagated up the nodes visited this iteration, and their own values updated accordingly (right). This process is repeated until a given computational *budget* is met, then the most successful root child – indicating the most successful action – is selected.

The most common form of MCTS is the *upper confidence bounds for trees* (UCT) algorithm [5], which is characterised by using an *upper confidence bounds* (UCB) calculation for child selection during tree descent. This calculation gives a good balance between *exploitation* and *exploration*, but the expansion strategy itself has a strong exploratory aspect, as all untried actions are expanded before any are retried.

UCT and Creative Search: UCT does not directly apply to bitstring domains, and is generally not applied to creative domains at all,[1] for a number of potential reasons:

1) *Repetition*: UCT is an optimising search that naturally converges to high-reward actions, and, being a form of reinforcement learning, benefits from repeated visits to high-reward states to maximise confidence in its ultimate choice. However, such repetition is anathema to creative search, in which every repetition is a wasted opportunity.

2) *Direction*: UCT applies to sequential decision processes, with decisions made closer to the root inevitably having greater influence on the search than those further down the tree. However, the reversible nature of bitstring mutation violates this assumption of sequentiality in this type of domain.[2]

3) *Root Selection*: UCT requires a root node, and will explore the search space reachable within D actions of the root for a search of depth D. The search will be strongly localised towards the root, and a poor initial choice of root node could be potentially disastrous.

[1]With the exception of Chevelu et al.'s UCT paraphrase generation [6]

[2]UCT has been applied to rooted, directed acyclic graphs [7], [8], but not (non-rooted, undirected, cyclic) graphs.

C. Hill Climbing

Hill climbing (HC) is another form of optimising search, in which local search is applied to incrementally improve candidate solutions. We implement a simple *iterated local search* (ILS) for comparison, in which the search is perturbed to a new location in the search space after each local maximum is reached. Hill climbing methods have some key fundamental similarities to the UCT method as applied here.

III. Upper Confidence Bounds for Graphs

We now propose a new variant of UCT called *upper confidence bounds for graphs* (UCG), for applying the basic approach to general graphs. The approach is similar in principle to *iterated local search* (ILS), exploiting the systematic expansion of UCT for local search.

A. Motivation

Given the limitations describe above, one may wonder why we wish to apply UCT to creative search. This is mainly because it offers one significant advantage not offered by evolutionary search: *systematic expansion*. Put simply, "mutation is random" [9, p. 41], whereas UCT exhaustively tries all available actions from a given state s before searching beyond s.

Consider Figure 3, which shows the probability of selecting a given bit b in a bitstring of length B=64, if a single bit is chosen each time the state is visited. Since UCT will try an untried action on each visit, the probability of selecting bit b will increase linearly to P=1.0 on the 64^{th} visit.

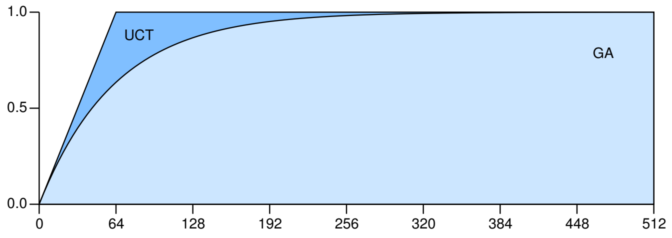

Fig. 3. Probability of selecting bit b in a size-64 bitstring, by visit count.

However, for a GA which randomly selects a bit each visit, the probability of hitting one specific bit in 64 visits is $P{\approx}0.635$, according to the following equation,[3] where b is the bit count and s is the visit count:

$$P(b,s) = \frac{\sum_{i=0}^{B}(-1)^i \binom{b}{i}(b-i)^s}{b!b^s} \quad (1)$$

While this difference may not seem that important, the situation becomes more striking when one considers that it is not known beforehand *which* of the 64 bits is most important – otherwise the search would be trivial – or even whether any bit is more important than the others. All 64 bits need to be tried, in order to guarantee that no key mutation is missed.

This situation is summarised in Figure 4, which shows the probability of all 64 bits having been selected if a single bit is chosen per visit. UCT is guaranteed to have tried all

[3]Based on a Stirling number of the second kind.

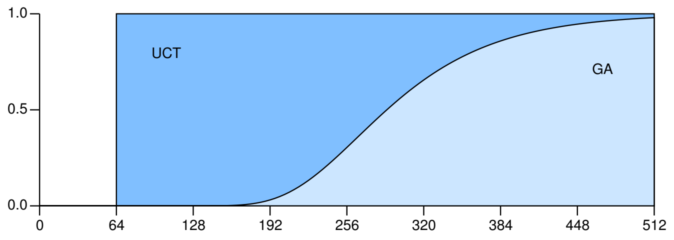

Fig. 4. Probability that all 64 bits have been selected, by visit count.

bits after 64 visits, for a probability of P=1.0. However, the probability that a GA would have tried all 64 bits after 64 visits is effectively zero ($P \approx 3.22 \times 10^{-27}$), from the equation:

$$P(b, s) = \frac{\sum_{i=0}^{b} -(-1)^{i+1} \binom{b}{i} (b-i)^s}{b^s} \tag{2}$$

As a general guideline, random selection requires around ten times as many visits as there are bits to achieve 95% certainty that all bits have been selected. But for domains of realistic complexity, it is unlikely that a given state will be selected for mating anywhere near this number of times.

As a concrete example of the implications of this point, consider the game Lammothm, which was produced by the LUDI system during an evolutionary search for combinatorial games [10]. Two players, White and Black, take turns placing a piece of their colour on an empty board cell, capture surrounded enemy pieces (as per Go), and win by connecting their sides of the board with a chain of their pieces.

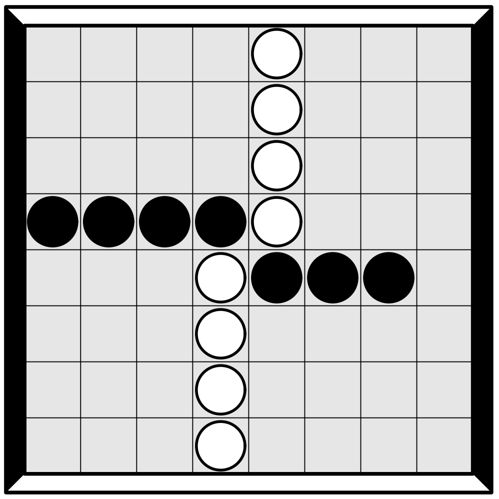

Fig. 5. White connects the white sides to win this game of Lammothm.

Lammothm's rule set includes the ludeme ($tiling\ square\ i\text{–}nbors$), which means that indirect neighbours (i.e. diagonals) count as being connected, hence White wins the game shown in Figure 5. Lammothm was deemed "playable" and fit enough to survive the filtering process, but was otherwise unremarkable; most games were an easy win for the first player (White) due to the presence of diagonal connections, and interesting situations rarely developed.

However, it was later realised that this mediocre rule set is only one mutation away from something much greater. If the $tiling$ rule is mutated to ($tiling\ square$), then the game suddenly becomes more interesting and almost equivalent to Gonnect, one of the best connection games of recent years [11]. Without diagonal connections, the position shown in Figure 5 would represent a temporary deadlock. The game would continue and become increasingly tense, as the board fills up and a player is eventually forced into a fatal move that makes one of their groups vulnerable to capture to resolve the game.

With evolutionary search there is no guarantee that this crucial mutation would ever be tried. With the systematic expansion of UCT, it is more likely that this mutation would be tried within a reasonable number of visits to this rule set.

B. Algorithm

The basic UCG approach is summarised in Listing 1. Graph G, the set of known states, starts empty and grows as the search progresses. Each vertex represents a unique state s within the graph, and each edge an action a that defines a symmetric transition between its two incident states. One edge is added per search iteration, hence the number of iterations at any point will be given by the number of edges $|E|$.

Algorithm 1 UCB for Graphs

```
1:  function UCG
2:      $G \leftarrow \emptyset$
3:      while $|E| < budget$
4:          if $rand(1) < 1/(|V|+1)$
5:              $G \leftarrow G$ + new $v$            /* add saltation */
6:          $v \leftarrow$ SELECT($G$)              /* select root */
7:          $v \leftarrow$ TRAVERSE($v$)          /* find most urgent */
8:          $v' \leftarrow$ EXPAND($v$)                /* expand it */
9:          UPDATE($v'$)                /* backpropagate value */
10: end function
```

Each iteration, a new vertex v may be added with probability:

$$P(v) = \frac{1}{|V|+1} \tag{3}$$

based on the number of vertices $|V|$. These constitute top-level nodes that represent *saltations* (i.e. jumps within the genotype space) denoted $\odot$.

The SELECT(G) function selects a vertex $\odot$ from the *saltation set* S to act as temporary root for this iteration. The TRAVERSE(v) function uses UCB to traverse the graph through adjacent steps from $\odot$ to its most urgent vertex v. Vertices are marked as "visited" as they are selected, and previously visited vertices are excluded from being revisited this iteration. The EXPAND(v) function selects an untried action a from v, and either creates a new vertex $f(v, a) \rightarrow v'$ or selects an existing vertex if v' already exists within the graph, and creates a new edge between v and v'. The UPDATE(v') function backpropagates the estimated value of v' back through the chain of visited vertices to the root $\odot$, updating the relevant reward and visit count statistics as per UCT. The result of the search is the highest valued vertex in G when the search budget is reached.

The return action a', which would revert v' to its original form $f(v, a') \rightarrow v$, is removed from v' to discourage revisits to previously known states. Maintaining a set S of temporary roots dilutes the impact of any individual root that performs poorly. The "visited" flags temporarily impart a direction for the duration of the current iteration, so the search will generally traverse away from the current root. All vertices expanded from a given saltation $\odot$ will form a connected set that includes $\odot$.

Figure 6 shows one iteration of the UCG "graph policy", on a graph under construction in a bitstring domain of size B=6. This graph currently has four saltations, one of which $\odot$ is selected as root for this iteration. $\odot$ is fully expanded, so traversal continues to its most urgent neighbour v_a, as shown. This node is also fully expanded, so traversal continues to v_a's most urgent neighbour v_b, as shown. This node has four untried actions to choose from, three of which will lead to known states and one which will produce a new vertex v'.

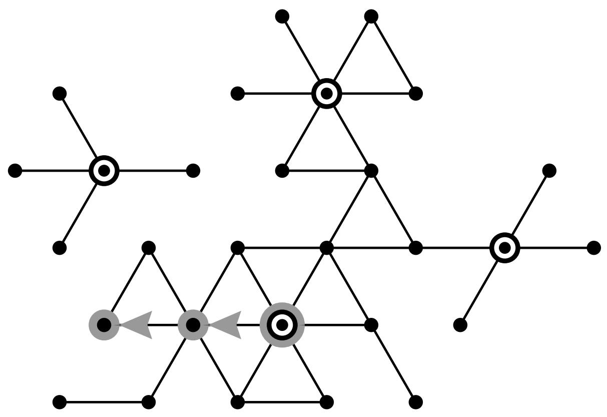
Fig. 6. One iteration of graph traversal from root $\odot$ to its most urgent node.

Note that three of the saltation groups connect form a single set in this example. Such a high degree of interconnectivity between saltations may be expected for simple domains such as B=6, but will not typically occur for more complex domains.

C. Implementation

The root node for each iteration $\odot$ may be selected randomly, or using UCB, although we found in the experiments that *tournament selection* (with $t = 4$) gives good results.

Each vertex maintains a *maximum* rather than *average* reward value as per standard UCT. This is generally more robust in sparse domains with high branching factors but low information, as is often the case in creative tasks, as averaging can drown out high-value outliers amongst the low-value noise. The only other known use of UCT for PCG also uses maximum reward values [6].

It is possible that a traversal can "paint itself into a corner" by visiting nodes in such an order that the current vertex has no expandable or unvisited neighbours, hence nowhere to step to. Such cases will be rare, but should not be ignored as they represent a trap state that may be repeated ad infinitum from this root $\odot$ without progress. The dead end vertex's reward value can be decreased by a small random amount, to discourage the repetition of this degenerate traversal route. Note that G will not otherwise have any terminal (leaf) nodes, as all states can be reached from all others.

Parallelisation: A separate hash table is maintained for each saltation $\odot$, for efficiency. This eliminates the possibility of interconnectivity between saltation groups, but this was found to not be an issue for realistically complex domains in which saltations groups are unlikely to reach each other anyway.

UCG therefore parallelises naturally to $|S|$ threads, although pre-calculating the saltation set S prior to the search loses the capacity to progressively customise new saltations to the developing situation. For example, new saltations may be crossed over from existing high-value states if quality is utmost, or to maximise the distance to the current population to encourage *creative leaps* [12] if novelty is utmost. New saltations are therefore added progressively in a single thread in the experiments.

IV. Experiments

We now compare the performance of MC, HC, GA and UCG on two creative test domains. The *fitness* $f(s)$ of a state s is given by the average of its typicality $t(s)$ and quality $q(s)$. The novelty of states is given by the diversity within the *elite* (top 10%) of the resulting population. A search is successful in this context if it produces a diverse population of fit states. The *search budget* is based on the number of *fitness function evaluations* (FFEs), as these measurements are typically the most computationally expensive part of any creative task.

A. Search Settings

The GA mutation rate is set to $P(m) = 1/B$ (with a guarantee of at least one mutation), hence multiple bits may be flipped per operation, increasing the potential for diversity. To achieve a similar effect in UCG, a number of untried actions n_a will sometimes be combined into a single action in the expansion step, that number given by:

$$n_a = |gaussian(0, \sqrt{B}/4)| + 1 \tag{4}$$

The GA crossover rate is set to 50% and local GA population size capped at 1,000. For UCG search, the UCB exploration parameter C is set to 0.125, biasing traversal towards exploitation; we rely on the distribution of saltations and systematic expansion for much of the exploration. The standard UCB1 formula [3] is used for child node selection:

$$\text{UCB1} = \overline{X}_j + C\sqrt{\frac{2\ln n}{n_j}}$$

HC denotes a simple hill climbing algorithm with iterated local search.

B. Diversity

Only those states with typicality $t(s) \geq 0.75$ are saved to the final population, as the user will usually only want to see reasonably typical examples of the domain. The genotypic diversity within the elite 10% of this final population is measured using Morrison and De Jong's moment of inertia method [13], which provides an efficient way to calculate the pair-wise Hamming distance of the member bitstrings.

Covering	Diversity
Random	0.224
Systematic	0.241

TABLE I. DIVERSITY FOR BITSTRINGS $i = 0..999,999$.

Table I highlights a curious property of this approach that emerged during the experiments: an exhaustive sampling of the members of a population (e.g. bitstring encodings of the integers $i = 0..999,999$) will be more diverse than a random sampling within this same range (regardless of order). An exhaustive covering will exercise every possible combination within its range, which already gives UCG a head-start in terms of diversity due to its systematic expansion policy.

We now describe two creative test domains, and the relative performance of MC, HC, GA and UCG search for each.

C. Biominoes

Biominoes are simple *polyomino*[4] shapes inspired by Dawkins' biologically-inspired biomorphs [9]. We consider biominoes that fit within an 8×8 grid, each cell of which maps directly to a bit within a bitstring of size B=64.

1) Fitness: Typicality $t(s)$ is measured as the ratio of connected on-pixels to total on-pixels, giving a smooth value function leading to $t(s) = 1.0$ for *complete* cases in which all on-pixels are members of a single connected set.

Quality $q(s)$ is the average of:

1) Horizontal symmetry (maximised).
2) Vertical symmetry (minimised).
3) Surface area (maximised).

Fitter states will therefore be complete polyominoes with strong horizontal symmetry, weak vertical symmetry, and many adjacent pixels of opposite parity. Figure 7 shows some typical examples produced during the experiments.

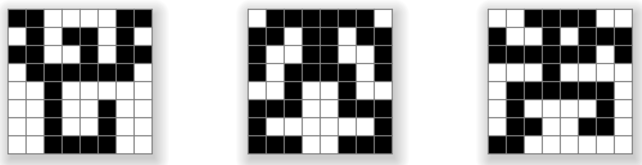

Fig. 7. Typical 8×8 Biominoes of quality 0.823, 0.771 and 0.734.

This domain is "well behaved" in the sense that:

1) There is a direct correspondence between the genotype and phenotype.
2) Genotype distance and phenotype distance are strongly correlated.
3) Every state gives a continuous and meaningful fitness value.
4) Any change to a genotype generally results in a change in fitness of proportionate scale.

[4]Orthogonally connected sets of squares [14].

2) Results and Discussion: Figure 8 shows the results of MC, HC, GA and UCG searches for 8×8 biominoes, for a budget of up to 1,024,000 FFEs, averaged over 100 runs. The plot shows the average mean fitness of the the top 10% of the current population for milestone points throughout the search, with the whiskers indicating the range (i.e. average minimum and maximum fitness values) about each mean.

GA and UCG both outperform MC and HC, and converge to similar mean fitness as the search budget is approached. However, UCG consistently produces the highest maximum fitness in most cases. HC is the worst performer in terms of fitness.

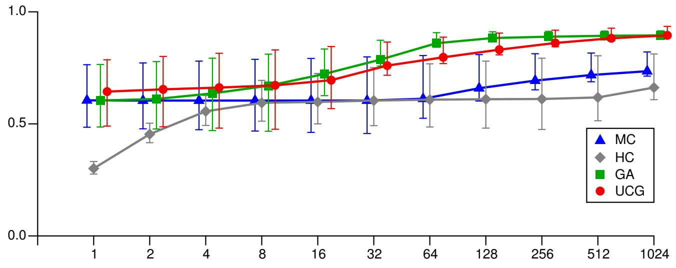

Fig. 8. Fitness (min/mean/max) by FFE count (×1000) for Biominoes.

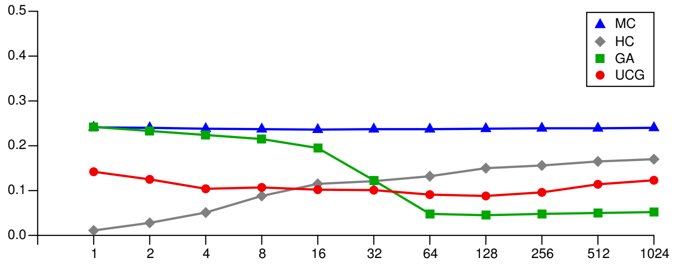

Fig. 9. Diversity by FFE count (×1000) for Biominoes.

Figure 9 shows the corresponding diversity within the top 10% of the current population at each milestone. Note that GA diversity drops from around the 8,000 mark while UCG diversity remains reasonably consistent throughout. UCG outperforms GA on diversity for that period in which GA outperforms UCG on fitness (≈32,000–128,000 measurements). HC shows poor diversity for low iteration counts (as it explores the local area of its starting point), but diversity increases as other points in the search space are considered.

D. Pentominoes

The Pentominoes domain involves packings of the twelve *pentominoes*, shown in Figure 10, in a 6×10 grid. The task is to find puzzle challenges consisting of placements of three *hint* tiles, such that the packing of the remaining tiles is:

1) unique,
2) deducible, and
3) interesting.

Each challenge is represented as a bitstring of size B=39, where the first four bits encode P_1 the index of the first hint piece, the next seven bits encode Pl_{l_1} the placement index for P_1, and so on.

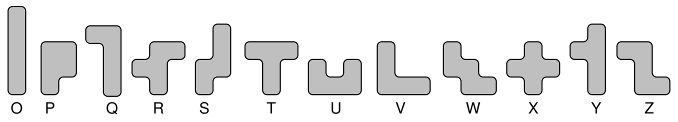

Fig. 10. The twelve pentominoes (Conway labelling).

1) Fitness: Typicality $t(s)$ is the ratio of remaining tiles that can be placed using level-1 *deductive search* [15]. A fully deducible challenge will score $t(i) = 1$, while any degenerate challenge with self-intersecting hints (which occurs in over 90% of randomly generated cases) will score 0 for both typicality and quality.

Quality $q(s)$ is based on the number of deductions that can be applied to s (as opposed to "obvious" moves), its computational cost, and a penalty for obvious starting moves. Fitter states will therefore be non-self-intersecting, have unique and fully deducible solutions, and be difficult within one deductive *level of embedding* (1-LoE).[5] For example, Figure 11 shows a challenge using the hint tiles S, U and Y, whose solution can be deduced using pure logic as shown.

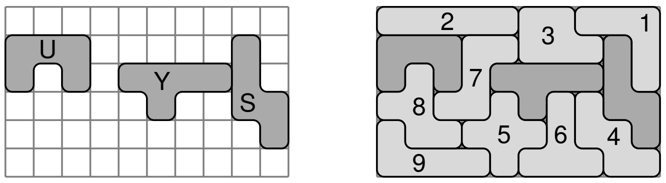

Fig. 11. An SUY challenge, and the sequence of deductions to solve it.

This is a more difficult domain to search as its design space is very sparse. Over 90% of randomly selected states will have a fitness of 0, yielding little information and giving the search little traction at low iteration counts. Further, there is a weaker correspondence between genotype distance and phenotype distance, as the genotype bits encode indices into lookup tables for piece placement rather than having any direct geometric relationship with the resulting placements on the board.

2) Results and Discussion: Figure 12 shows the mean fitness at various milestones during MC, HC, GA and UCG searches of the Pentominoes domain, up to a budget of 10,240 measurements, averaged over ten runs. The search budget is smaller for the Pentominoes domain than the Biominoes domain, as the fitness measurement is much more expensive. Figure 13 shows the corresponding diversity measurements.

No search significantly outperforms the baseline MC search for this more difficult domain, but surprisingly GA actually performs *worse* than random selection at most milestones, both in terms of fitness and diversity, especially for larger search budgets. We understand this to be an effect of not allowing duplicates in the master population, so as GA search converges to a local maximum it will produce many similar duplicates, most of which will already exist in the master population and hence be discarded. The only new states to be added will be those further from the local maximum, which are likely to have lower fitness. UCG does not appear to suffer this effect, at least not to any noticeable extent.

[5]"1-LoE deduction" means that a challenge can be solved using deduction without recursion, so should not be too difficult for human players [15].

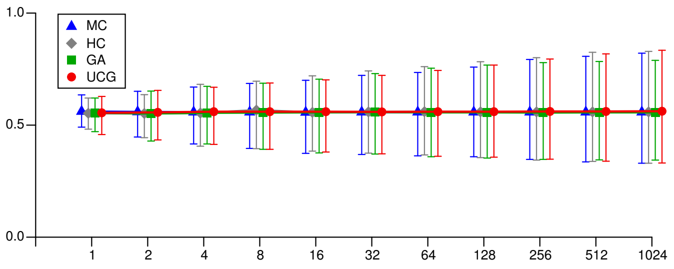

Fig. 12. Fitness (min/mean/max) by FFE count (×10) for Pentominoes.

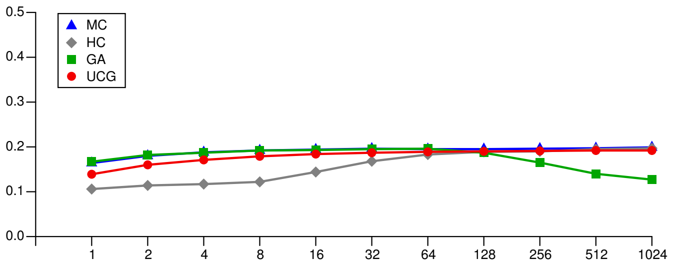

Fig. 13. Diversity by FFE count (×10) for Pentominoes.

E. Expression Trees

The highly structured nature of the Pentominoes genotype is perhaps more suited to an *expression tree* than a bitstring. For example, Figure 14 shows a natural representation of hint set H as three dependent pieces, each with a dependent placement. We now compare UCT-based and evolutionary searches for this alternative representation of the Pentominoes domain. Note that the task and fitness measurements are the same, only the genotype representation has been changed.

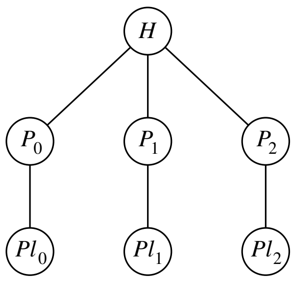

Fig. 14. Expression tree representation of the Pentominoes domain.

In order to accommodate this new representation, GA is replaced with a standard *genetic programming* (GP) search [16], in which pieces and placements are crossed over and mutated as a discrete whole, for greater *modularity*.

Similarly, UCG is replaced with a standard UCT search, in which the expression tree is "unrolled" to give a sequence of decisions $\{P_1, Pl_{l_1}, P_2, Pl_{l_2}, P_3, Pl_{l_3}\}$ and the simulation step involves randomly selecting values within range for pieces and placements that have not been chosen yet. UCB1 is again used as the child selection policy, with the exploration constant C set to 0.125.

1) Results and Discussion: Figures 15 and 16 show the results for mean fitness and diversity, respectively.

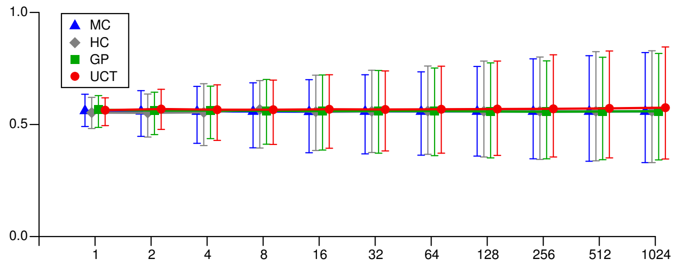

Fig. 15. Fitness (min/mean/max) by FFE count (×10) for Pentominoes (treed).

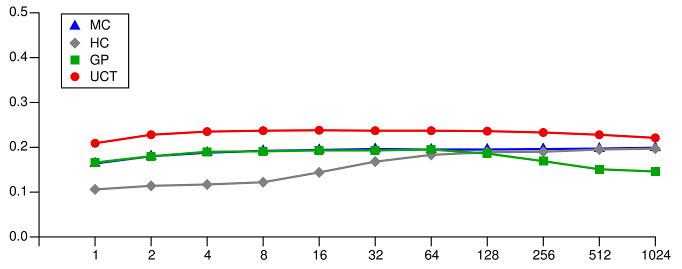

Fig. 16. Diversity by FFE count (×10) for Pentominoes (treed).

We see a slight improvement in the both GP and UCT searches over GA and UCG in terms of fitness. However, we see a more striking improvement in diversity of UCT over all other methods in this case, including MC. We believe that this is due to the exhaustive nature of UCT search. We note that Cazenaze reports superior performance with the related method of *nested Monte Carlo* (NMC) search over GP search for expression tree discovery [17].

F. Timings

Table II shows the average time of each experimental run, in seconds. All experiments were run on the single thread of a standard Macbook laptop with $i5$ processor. The UCT-based approaches are slower, indicating the computational overhead of performing multiple UCB calls during graph traversal.

Search	**Biominoes**	**Pentominoes**
MC	0.035ms	3.853ms
HC	0.033ms	3.587ms
GA	0.081ms	2.843ms
UCG	0.159ms	5.222ms
GP	–	3.539ms
UCT	–	5.909ms

TABLE II. INDICATIVE TIMINGS PER FFE.

V. SUMMARY

Table III summarises the results for both domains at milestone 1,024 (i.e. at the conclusion of each search), with 95% confidence intervals.

For the (simpler) Biominoes domain, GA and UCG clearly outperform the other search methods in terms of quality, and of these two UCG clearly outperforms GA in terms of diversity.

Search	**Biominoes**		**Pentominoes**	
	Quality	Diversity	Quality	Diversity
MC	0.821 ±0.007	0.240 ±0.001	0.821 ±0.025	0.199 ±0.003
HC	0.812 ±0.017	0.170 ±0.046	0.829 ±0.018	0.197 ±0.004
GA	0.915 ±0.007	0.052 ±0.005	0.789 ±0.029	0.127 ±0.005
UCG	0.935 ±0.019	0.123 ±0.055	0.834 ±0.019	0.192 ±0.005
GP	–	–	0.817 ±0.031	0.146 ±0.003
UCT	–	–	0.846 ±0.032	0.221 ±0.005

TABLE III. MAXIMUM QUALITY AND DIVERSITY PER SEARCH.

For the (more difficult) Pentominoes domain, UCG and UCT perform much better than GA and GP and slightly better than MC and HC in terms of quality, and perform much better than GA and GP in terms of diversity. The UCT-based methods (UCG and UCT) generally outperform the other methods in terms of maximum quality, although the margins are slight and not always significantly clear.

VI. CONCLUSION

The UCG method combines the systematic expansion of UCT with the balanced action selection of UCB, to produce a search method for arbitrary graphs that encourages diversity as well as fitness, which are both important factors in creative search. Its performance is compared to MC, HC and GA searches for two test domains of varying difficulty.

The UCT-based methods generally outperformed the other methods in terms of maximum quality, although the margins were slight and not always significantly clear. In the (simpler) Biominoes domain, GA and UCG yielded better quality individuals, while UCG yielded higher diversity than GA. In the (more difficult) Pentominoes domain, GA and GP actually performed worse that random, while the UCT-based methods performed better than random, in terms of both quality and diversity.

These results suggest that UCT-based methods may exhibit behaviours conducive to creative search, where both quality *and* diversity are required, and warrant further investigation in this context. Future work might include a comparison with more complex search types, and deeper investigation into UCT for expression tree search in creative domains.

ACKNOWLEDGMENT

Thanks to Néstor Romeral Andrés for help deriving equations 1 and 2. This work was funded by EPSRC grant EP/I001964/1 *UCT for Games and Beyond*.

REFERENCES

[1] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, “Search-based Procedural Content Generation: A Taxonomy and Survey,” *IEEE Trans. Comp. Intell. AI Games*, vol. 3, pp. 172–186, 2011.

[2] G. Ritchie, “Some Empirical Criteria for Attributing Creativity to a Computer Program,” *Minds & Machines*, vol. 17, pp. 67–99, 2007.

[3] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A Survey of Monte Carlo Tree Search Methods,” *IEEE Trans. Comp. Intell. AI Games*, vol. 4:1, pp. 1–43, 2012.

[4] E. Den Heijer and A. E. Eiben, “Maintaining Population Diversity in Evolutionary Art,” in *Proc. Evol. Biolog. Inspired Music, Sound, Art and Design (EvoMUSART)*, Malaga, 2012, pp. 60–71.

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

[6] J. Chevelu, T. Lavergne, Y. Lepage, and T. Moudenc, "Introduction of a new paraphrase generation tool based on Monte-Carlo sampling," in *Proc. 4th Int. Joint Conf. Natur. Lang. Process.*, vol. 2, Singapore, 2009, pp. 249–252.

[7] A. Saffidine, T. Cazenave, and J. Méhat, "UCD: Upper Confidence bound for rooted Directed acyclic graphs," in *Proc. Conf. Tech. Applicat. Artif. Intell.*, Hsinchu City, Taiwan, 2010, pp. 467–473. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/TAAI.2010.79

[8] F. de Mesmay, A. Rimmel, Y. Voronenko, and M. Püschel, "Bandit-Based Optimization on Graphs with Application to Library Performance Tuning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, Montreal, Canada, 2009, pp. 729–736. [Online]. Available: http://portal.acm.org/citation.cfm?id=1553468

[9] R. Dawkins, *The Blind Watchmaker*. London, UK: Penguin, 1986.

[10] C. Browne, *Evolutionary Game Design*. Berlin: Springer, 2011.

[11] ——, *Connection Games: Variations on a Theme*. Natick, Massachusetts: AK Peters, 2005.

[12] L. Noy, Y. Hart, N. Andrew, O. Ramote, A. Mayo, and U. Alon, "A quantitative study of creative leaps," in *Proc. Third Int. Conf. Comput. Creativity*, Dublin, 2012, pp. 72–76.

[13] R. W. Morrison and K. A. D. Jong, "Measurement of population diversity," in *In 5th International Conference EA 2001, volume 2310 of lncs*. Springer, 2002, pp. 31–41.

[14] S. W. Golomb, *Polyominoes*. George Allen & Unwin Ltd, 1965.

[15] C. Browne, "Deductive Search for Logic Puzzles," in *Proc. Comput. Intell. & Games*. Niagara Falls: IEEE, 2013, pp. 1–8.

[16] J. Koza, *Genetic Programming*. Massachusetts: MIT Press, 1992.

[17] T. Cazenave, "Monte-Carlo Expression Discovery," *Int. J. Artif. Intell. Tools*, vol. 22, pp. 1–21, 2013.