Creativity as Search for Small, Interesting Programs

Dan Ventura
Computer Science Department
Brigham Young University
ventura@cs.byu.edu

Daniel G. Brown
David R. Cheriton School of Computer Science
University of Waterloo
dan.brown@uwaterloo.ca

Abstract

We explore creativity and search, particularly focusing on discovering novel, valuable artifacts. Using a density-based argument, we show it is generally infeasible for base-level search mechanisms to discover such artifacts in any but simplistic domains, As such, creativity cannot be implemented (directly) as search. We then appeal to algorithmic information theory to argue that one solution to this conundrum is to instead consider creativity as a search for programs that produce novel, valuable artifacts that have short descriptions but require long run times.

Introduction

Search is "the" metaphorical abstraction of computational creativity. Boden's early work introduced the high-level idea (1991), and Wiggins' seminal work (2006) on his creative systems framework (CSF) solidified the metaphor as a foundational component of the field. Since then, creativity-as-search has proven its value by shaping research and providing a common framework for research on disparate application domains. Wiggins' work is formal and precise. However, the account is abstract; it does not operationalize the idea, nor has any subsequent. We explore a difficulty with creativity-as-search and offer some ways to avoid this difficulty, perhaps shedding further light on the search ideaand how it should be applied to computational creativity.

In particular, we suggest that the CSF applied to base-level artifacts is perhaps doing a disservice as a metaphor for creativity in the same way that the trope of the room full of typewriting monkeys suggests that Shakespeare was just lucky. We explore simple domain examples that seem to validate the idea of creativity-as-search. We then scale up our examples to more real-world creative tasks, to expose the difficulty of applying the search metaphor: *due to search space size, base-level search over objects cannot be the creative process.* We describe creativity as a "hack" for navigating computational spaces too large for efficient search. Another way to think about this is that if search plays a part in creativity, it must be augmented to make the space manageable. We suggest combining creativity-as-search with algorithmic information theory concepts. Perhaps the monkeys are not themselves typing randomly but are executing a creative program. Some programs themselves can search efficiently for valuable outputs. We make the case that *creativity isn't a direct search for artifacts, but it is instead a search for small programs that produce them.* Searching for small programs has the benefit of both making the search space manageable and producing programs that exhibit properties associated with creativity (Mondol and Brown 2021a).

The Trouble with Creativity as Search

Let $\mathcal{B}$ be all possible artifacts in a domain of interest and $\mathcal{G} \subset \mathcal{B}$ be a set of goal artifacts. The creative process can be thought of as the iterative application (starting from the empty set) of a traversal function $T : \{2^{\mathcal{B}} \rightarrow \mathcal{B}\}$ until the traversal produces an artifact $g \in \mathcal{G}$. T defines a search strategy, and we are interested in strategies that find at least one $g \in \mathcal{G}$.

We might characterize the likelihood of finding such a g, over the set $\mathcal{T}$ of all possible search strategies, as the density $d = \frac{|\mathcal{G}|}{|\mathcal{B}|}$. If $d \gg 0$, there are many good "solutions" to the search, and many search strategies will quickly succeed in discovering one of them. Under such conditions creative search is not necessary. When $d \ll 1$, most search strategies will not identify any element of $\mathcal{G}$ in a reasonable amount of time—creativity becomes critical (or, put another way, any successful strategy appears creative). As we proceed, we will make use of this density argument as we analyze several example domains. This abstraction is not much different from optimization, where "uncreative" objects have score zero and "creative" objects have score 1. Optimizing the score function is here exactly equivalent to searching for creative objects. In this formulation, aesthetics in optimization algorithms would directly map onto creativity search algorithms (and vice versa).

Can creativity be thought of as a "hack" for effectively searching very large spaces? What is the relationship between heuristics and creativity, and between brute-force search and non-creativity? We begin by considering some position-strategy games, where we assign value to game statuses in which a particular player is the winner.

First, consider tic-tac-toe. The game has an upper bound of $3^9 = 19683$ board states ($\mathcal{B}$); this number can be lowered to just 765 due to various symmetries, constraints and redundancies. Of those, 138 are terminal states: 91 a win for the starting player; 44 a win for the other player; and 3 a draw. Assuming the goal is not losing, $d = \frac{\mathcal{G}}{\mathcal{B}} = \frac{94}{765} \approx 0.12$ for the starting player and $d = \frac{\mathcal{G}}{\mathcal{B}} = \frac{47}{765} \approx 0.06$ for the other player. Placing marks randomly gives the starting player a 12% chance of a draw or better, *regardless of the strategy being employed by the other player.* Of course, a space of 765 artifacts is trivial for computational brute force search, so a search algorithm makes sense here: for each board state, one can identify a play that will push the game toward the player's best outcome. However, what makes the game solved, and eliminates any possibility/need for creativity is that those 765 states have been abstracted into a set of eight ordered rules guaranteeing no worse than a draw. At best, creativity in tic-tac-toe would be a strategy that aims at a surprising outcom.

A game like checkers is still simple, but it's much more complex than tic-tac-toe, with a naive upper bound $|\mathcal{B}| \leq 5^{32}$ (for the traditional 8×8 board, where each position can be either empty, held by a regular piece, or held by a kinged piece), which again can be tightened by application of symmetry, constraint and redundancy arguments to around 5×10^{20} positions. Like, tic-tac-toe, the game of checkers has been (weakly) solved by the Chinook algorithm (Schaeffer et al. 2007), and like tic-tac-toe, perfect play from both players will always result in a draw. However, unlike tic-tac-toe, this solution is beyond the abilities of human players, so the game lives in the interesting limbo of disallowing computational creativity (because searching for a good outcome $g \in \mathcal{G}$ is "trivial" computationally by using the Chinook algorithm) while potentially still admitting human creativity (because that computational solution is incomprehensible to human players). Again, we could imagine a pattern in which a human player either did not play optimally, or in which the human player (perhaps incorrectly) attempted to abstract away patterns from watching Chinook's play in actual (or contrived) examples; such a search algorithm might be faster or shorter than the one built into Chinook, but still have an aesthetic value to it, by delighting observers with its cleverness. Still, the entire Chinook algorithm could be represented as a lookup table to identify for each position what the next step should be, much as with tic-tac-toe; while this would be quite large, smaller versions than 8x8 checkers could easily be stored in full on a single computer.

The game of Go is yet far more complex, and cannot be brute forced by a computer; indeed, when AlphaGo played Lee Sedol in 2016, Sedol attributed creativity to the system because he was surprised by novel and useful choices that the algorithm made.

Spendlove and Brown propose a continuous spectrum from games to non-games as creative tasks (2023), and this density argument applies naturally across that spectrum. As a first example of a non-game "creative" task, consider identifying 3-letter English words. An upper bound on the number of such words is $|\mathcal{B}| = 26^3 = 17576$. Because a search strategy T for the space $\mathcal{B}$ induces an ordering on $\mathcal{B}$, there exist 17576! unique strategies for searching for a valid word. For example, one strategy simply moves to the lexicographically next three-letter string, starting with "aaa"; this process will terminate with the word "aah" in eight steps. However, according to the Scrabble dictionary, there are $|\mathcal{G}| = 1340$ 3-letter words in the English language. Calculating the density, $d = \frac{1340}{17576} \approx 0.076$ reveals that choosing three letters randomly will produce a valid English word 7.6% of the time. Also, a randomly chosen search strategy has a 7.6% chance of returning a valid English word as its first output.

Assuming some practical horizon h of generated artifacts, the probability of this random search process failing to produce a valid English word within that horizon is only 0.924^h. In other words, the proverbial monkeys with their typewriters will have no problem producing valid 3-letter English words.

Searching for good words

We can consider this word-search algorithm in a variety of fashions, which elucidate ways in which the CSF might or might not identify creative discovery methods.

One way to think of this search problem is as over a 3-dimensional space where each axis is a number line with 26 non-ordinal, discrete values. Then there are $(26!)^3$ different ways to order the axis labels that characterize the space. Do any of these orderings cluster all (or many of) the valid words together? And, if so, how can those orderings be discovered? Of course, in the case of this simple example, it has, like tic-tac-toe, already been completely solved, and can be conveniently represented as a dictionary tree that efficiently represents a (pruned) brute force search of the space: the tree has height 3 and branching factor 26, with pruned branches for illegal combinations and 1340 leaf nodes.

Another way to think about representing this space is as a joint probability distribution over 3-letter words $w = s_1 s_2 s_3$. This distribution can be factored as $p(w) = p(s_1)p(s_2|s_1)p(s_3|s_1, s_2)$, and, given enough data, these distributions can be modeled so that sampling the joint produces a valid word—in other words we can build a model that accurately models within-distribution words, which in this case means all valid 3-letter words. Again, with this simple example, the creativity-as-search mechanism seems justifiable, but we're working in a very small space.

We now increase the scale and complexity significantly by examining the *Library of Babel*, the subject of an influential short story by Luis Borges (1944). In that library exist all books that are exactly 410 pages long, with 40 lines per page and 80 characters per line. Defined this way, the domain of books $\mathcal{B}$ is finite, with a size of $|\mathcal{B}| = 29^{410\times40\times80} = 29^{1312000}$ (including spaces, commas and periods in addition to letters). The set $\mathcal{G}$ of novel and valuable books is a tiny subset of $\mathcal{B}$. As a very conservative approximation, let us consider G to contain all books that contain only valid English words (not necessarily in an any grammatically correct order, never mind in any kind of interesting/creative/good order). The average word length in English is 4.7 characters. There are 12972 five-letter words in English. An average effective sentence is typically not longer than 20 words in length, so roughly 120 characters. That means on average a book in the library would have 10933 sentences. 5-letter word density can be estimated as $d_w = \frac{|\mathcal{G}_w|}{|\mathcal{B}_w|} = \frac{12972}{26^5} = \frac{12972}{11881376} \approx 0.001$. Then sentence density can be estimated as $d_s \approx 0.001^{20} = 10^{-60}$, and then finally book density as $d_b \approx (10^{-60})^{10933} = 10^{-655980}$, which is an astoundingly small number, effectively 0. So, given any practical (or even any impractical) horizon h, the chance of a random search process failing to produce such a coherent book (never mind an interesting one) is effectively 100%. In other words, the proverbial monkeys may have been given way too much credit in the general scheme of things; further, having been given one book, there is no help given in identifying the next one.

This is akin to indexing or cataloguing the library—if a search strategy doesn't repeat books, it is a permutation of the library, which we explore in order until we get to a good book. This approach again shows the odd challenge of the search approach—how do we identify the *next* book to consider, as a function of the failures we've already detected? Because there are $|\mathcal{B}|!$ unique orderings, there are then also $|\mathcal{B}|!$ unique search strategies. Some find all of the elements in $\mathcal{G}$ before finding any in $\mathcal{B} - \mathcal{G}$. Another set will find any element in $\mathcal{G}$ only after finding every element not in $\mathcal{G}$.

If, instead, we think about building books using valid English words as our atomic symbols, we eliminate a huge number of nonsense books. This reduces the dimensionality of the space significantly, though it is still huge. Building a "dictionary" tree for all possible books, requires a brute-force enumeration of all possible books, whether using letters or words as our symbol basis, which is still completely infeasible.

Finally we can consider the statistical modeling approach, and both letter-based $p(b) = p(s_1)p(s_2|s_1)\ldots p(s_n|s_1,\ldots,s_{n-1})$ and word-based models $p(b) = p(w_1)p(w_2|w_1)\ldots p(w_m|w_1,\ldots,w_{m-1})$ have been built and are surprisingly good at producing coherent English language (or any other language, including mathematical and programming languages), given enough training data. Modern transformer-based models use this approach, though typically compromising between the simpler letter-based and more complex word-based by employing instead a symbol vocabulary of $\sim 50,000$ subword tokens $p(b) = p(t_1)p(t_2|t_1)\ldots p(t_k|t_1,\ldots,t_{k-1})$. This approach models patterns in the training data, and LLMs are admittedly impressive on some level. However, by the very nature of the data and the modeling process, they are not modeling creative output but instead learning exactly the opposite—to model the distribution of the training data. Indeed, it seems likely that we can make the argument that it is actually not possible to model creativity in this way because a) any and all training data fed to such models is not creative by definition—it is Ritchie's inspiring set $\mathcal{I}$ (2007) and b) the training objective of such models is to produce a smooth (latent) manifold such that interpolating between points produces some (linear) combination of those points. As a result, it is likely that language modeling is a creative dead-end that will itself never produce creative output.

On Searching for Programs, not Objects

One approach to bring us to a better place is to focus on the algorithm itself, rather than the items it finds, as the focus of creativity. This approach situates creativity at the Producer/Process levels in the 4 P's framework (Rhodes 1961; Jordanous 2016), rather than at the Products. But there is still a challenge: the set of valid search programs itself has good and bad elements, and we will still have to consider searching over those as well; the challenge of computational metacreativity is never far away.

Nonetheless, we consider this task of finding search algorithms. Consider the set $\mathcal{M}$ of all valid Turing machines that halt on all inputs and, when given a digital representation of a sequence or set of members of the set $\mathcal{B}$ as input, return a member of $\mathcal{B}$ as output. Each describes a search process. The sequence or set of members of $\mathcal{B}$ is the prior search items, or an inspiring set.

Some are useless and trivial (for example, a machine M which immediately returns the first object of $\mathcal{B}$ in its input (except to supply some trivial member of $\mathcal{B}$ on empty input), represents a version of the identity function, and never finds anything new. Some might well *appear* complex, but can still be trivial, since it is undecidable whether a given Turing machine M computes the identity function. Still others will perform the search strategies described in the previous sections.

Many never explore all members of $\mathcal{B}$, even approximately. If $\mathcal{B}$ is uncountably infinite, then any algorithm M will only explore a countable set of its members, but even for finite domains, the algorithm might loop back to an earlier example. As such, one restriction on valid members of $\mathcal{M}$ might be that for every sequence or set s, $M(s)$ outputs a member of $\mathcal{B}$ that is not found in s. Here, for any finite set $\mathcal{B}$, the entire set $\mathcal{B}$ will be enumerated in some order by M, including when the algorithm is initialized with the empty sequence. For finite sets $\mathcal{B}$, algorithms of this sort compute a permutation of $\mathcal{B}$. Regardless, the set of valid *functions* being computed by members of $\mathcal{M}$ is finite.[1]

[1]If $\mathcal{B}$ is infinite, then the set of valid functions is countably infinite, since each is computed by a Turing machine,

We can compare two different Turing machines that are equivalent to each other that come from this abstraction. For finite sets $\mathcal{B}$, the restriction that we've made that the machines halt on all valid inputs means that many uncomputability issues for arbitrary machines don't apply (they all terminate in finite time having enumerated all of $\mathcal{B}$). But we can still consider the number of iterations of the algorithm that are needed, starting with the empty history, before the algorithm discovers a good object. Or we can consider the total runtime (in the sense of steps of Turing machine calculation) by all iterations of the algorithm before it discovers a good object. Or we could consider the length of an encoding of the Turing machine program M that is used; in particular, if we view the machine M as a program written in a modern programming language, then the shortest program that computes the same function as M will have length $K_{\mathcal{M}}(M)$, where the function K is the Kolmogorov complexity function and the $\mathcal{M}$ subscript indicates the restriction to machines that halt on all inputs and otherwise satisfy our restrictions.

In a recent sequence of papers, Mondol and Brown explore computational aesthetics based on algorithmic information theory that starts from this set of comparisons (2021b; 2021a; 2021). They claim that creativity (or, at least, value) is witnessed primarily in short-length algorithms that take a long while to compute objects that are not (also) computable by short-running, short-length algorithms. That is, if an object is computable by a short, fast algorithm, it is trivial; if it is computable only by a long, fast algorithm, it is random. Objects whose only short-length algorithms are slow are interesting, as each bit of them requires substantial work to discover.

We can imagine two points in the CSF where the AIT framework comes to mind: the first is the actual process of choosing new objects, and the second is the choosing of the search algorithm in the first place. If the algorithm for choosing next attempts is a good one, then instead of searching aimlessly through the conceptual space, or requiring a giant look-up table to enable the search process, it will instead focus the search on cleverly moving towards a good outcome. In this sense, a high-quality creative search algorithm (perhaps unsurprisingly) focuses on compressible properties about the items already seen, and uses them to identify "next choices" from the set $\mathcal{B}$ of possible items.

The second place where the AIT frame comes to mind is in *identifying* such algorithms in the first place—in particular, a good search algorithm ought, itself, to be a short algorithm that, nonetheless, finds items that require substantial discovery time. One example of such an algorithm might be an enumerator that then applies a substantially complex-to-run evaluator to the enumerated objects, and which takes a while to make the decision of what to enumerate next. But we can imagine the metacreative task of enumerating over creative algorithms to find a good one to itself be a creative task, where the CSF is being applied to a set of artifacts that are themselves creative search algorithms—one needs a short, yet potentially slow, procedure to enable this search over algorithms. Which suggests a need for (at least) two levels of evaluation: E_b for base-level artifacts (may be considered part of a search algorithm) and E_a for search algorithms. Finding an E_a that successfully identifies short, slow algorithms is key to the meta-level search and may prove challenging. In addition, Ventura suggests that base-level evaluation is unlikely to be computable (Mondol and Brown's suggestions are all undecidable) and based on this premise gives a high-level proof sketch (reducing from the Halting Problem) for why computational creativity is not computable in the strong sense, independent of any difficulties (or lack thereof) due to meta-level recursion or issues with the size of the search space (2014).

Discovering one high-quality object x enables discovery of a huge set (all those objects $y \in \mathcal{B}$ whose conditional Kolmogorov complexity $K(y|x)$ is small) of other high-quality objects. Since x has no short, fast-running programs, neither do any of these choices of y. Mondol and Brown do give a definition of novelty using AIT, which looks at $K(y|B)$, where B is the previous members of $\mathcal{B}$ that have been produced; this measures how much a newly-found object differs from the ones already identified. A creative search algorithm can't just milk a high-quality discovery for all it's worth: it also keeps exploring, as once a creative x is found, all similar objects y are not novel. This finding links back to our claim that probabilistic language modeling is a creative dead end: re-shuffling training distributions cannot yield novel, valuable results.

In this sense, then, the creative search framework misses the creativity that sits in the actual creative search algorithm itself—how do we discover such an algorithm, and is this the real locus of creativity?

Conclusion

While creativity-as-search has been helpful to the CC community as an abstraction that facilitates discourse on core creativity ideas that are (hopefully) common across many application domains, it may also be somewhat misleading if taken literally. Our density-based analysis shows the general infeasibility of searching even moderately sized domains for novel, useful base-level artifacts. Instead, the search should be for short programs that produce the base-level artifacts of interest. In addition to still leveraging all the benefits of the creativity-as-search framework, this idea has at least two additional benefits: the search space may be small enough to make a search feasible, and searching for programs rather than base-level artifacts further emphasizes the common basis for creativity across domains, which are differentiated by what those programs do.

and there are countably many Turing machines; we note that the number of orderings of entries of $\mathcal{B}$ is uncountably infinite, so the space of valid orderings computed by Turing machines is a vanishingly small fraction.

Acknowledgments

We are grateful to Brad Spendlove for helpful conversations about this project.

The research of D.G.B. is supported by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada.

References

Boden, M. A. 1991. *The Creative Mind: Myths and Mechanisms.* London and New York: Routledge.

Borges, J. L. 1944. La biblioteca de Babel. In *Ficciones.* Editorial Sur.

Brown, D. G., and Mondol, T. 2021. On the problem of small objects. *Entropy* 23(11):1524.

Jordanous, A. 2016. Four PPPPerspectives on computational creativity in theory and in practice. *Connection Science* 28(2):194–216.

Mondol, T., and Brown, D. 2021a. Incorporating algorithmic information theory into fundamental concepts of computational creativity. In *Proceedings of the International Conference on Computational Creativity*, 173–181.

Mondol, T., and Brown, D. G. 2021b. Computational creativity and aesthetics with algorithmic information theory. *Entropy* 23(12):1654.

Rhodes, M. 1961. An analysis of creativity. *Phi Delta Kappan* 42(7):307–309.

Ritchie, G. 2007. Some empirical criteria for attributing creativity to a computer program. *Minds and Machines* 17:76–99.

Schaeffer, J.; Burch, N.; Björnsson, Y.; Kishimoto, A.; Müller, M.; Lake, R.; Lu, P.; and Sutphen, S. 2007. Checkers is solved. *Science* 317(5844):1518–1522.

Spendlove, B., and Brown, D. 2023. What makes gameplay creative? In *Proceedings of the 14th International Conference on Computational Creativity*, 98–101. Association for Computational Creativity.

Ventura, D. 2014. Can a computer be lucky? and other ridiculous questions posed by computational creativity. In *Proceedings of the Seventh Conference on Artificial General Intelligence*, 208–2017.

Wiggins, G. A. 2006. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge Based Systems* 19:449–458.