Using a large language model to generate program mutations for a genetic algorithm to search for solutions to combinatorial problems: Review of (Romera-Paredes et al., 2023).

Ernest Davis
Dept. of Computer Science
New York University
New York, NY 10012
davise@cs.nyu.edu

January 7, 2024

Abstract

FunSearch (Romera-Paredes et al., 2023) uses a large language model (LLM) of a component of an AI system that has generated some difficult-to-find solutions to combinatorial problems that are larger than was previously known. However, the excitement that has greeted this has been unwarranted. First, as compared to other applications of AI to mathematical research, FunSearch does not seem particularly promising as a technique and its contribution to mathematics is not unusually great. Second, FunSearch uses the LLM as a subroutine in a genetic programming algorithm, to generate mutations of one particular subroutine in a larger program. The LLM is not told either what problem is being addressed or what the overall program is. Thus, as compared to other application of LLMs to mathematics, the LLM in FunSearch is remarkable for the *shallowness* of the mathematical understanding that it seems to exhibit.

1 Introduction

FunSearch (Romera-Paredes et al., 2023) uses a large language model (LLM) of a component of an AI system that has generated some difficult-to-find solutions to combinatorial problems that are larger than was previously known. These results are (to the very limited degree that I can judge) moderately interesting mathematically, and FunSearch is likewise moderately interesting as an original approach to the use of AI for mathematical research. One aspect of the interaction between FunSearch and the human mathematicians involved in the project, discussed below, is indeed quite noteworthy and, as far as I know, original.

Inevitably, however, the PR machine at Deep Mind has announced this new system with blaring trumpets, and the technology media has been happy to join the parade. As reported in *Technology Review* (Heaven, 2023):

> Google DeepMind has used a large language model to crack a famous unsolved problem in pure mathematics. In a paper published in *Nature* today, the researchers say it is the first time a large language model has been used to discover a solution to a long-standing scientific puzzle-producing verifiable and valuable new information that did not previously exist. “It’s not in the training data—it wasn’t even known,” says coauthor Pushmeet Kohli, vice president of research at Google DeepMind.

> Large language models have a reputation for making things up, not for providing new facts. Google DeepMind's new tool, called FunSearch, could change that. It shows that they can indeed make discoveries if they are coaxed just so, and if you throw out the majority of what they come up with.

Nature's own news item about the article (Castelvecchi, 2023) is entitled "DeepMind AI outdoes human mathematicians on unsolved problem". It quotes Pushmeet Kohli further as saying, "This is the first time anyone has shown that an LLM-based system can go beyond what was known by mathematicians and computer scientists. It's not just novel, it's more effective than anything else that exists today."

As we will see, some of this is doubtfully true; some of it is misleading; some of it is irrelevant; and none of it is nearly as impressive as it sounds.

A DeepMind blog (Fawzi and Romera-Paredes, 2023) made even more extravagant predictions:

> FunSearch demonstrates that if we safeguard against LLMs hallucinations, the power of these models can be harnessed not only to produce new mathematical discoveries, but also to reveal potentially impactful solutions to important real-world problems.
>
> We envision that for many problems in science and industry - longstanding or new - generating effective and tailored algorithms using LLM-driven approaches will become common practice.
>
> Indeed, this is just the beginning. FunSearch will improve as a natural consequence of the wider progress of LLMs, and we will also be working to broaden its capabilities to address a variety of society's pressing scientific and engineering challenges.

The Twitterverse is likewise in a tizzy; Stanford Professor Eric Brynjolffson tweeted, "This is undoubtedly just the beginning of a new age of scientific discovery."

I will argue below that:

- As compared to other applications of AI, or more broadly computer technology, to mathematical research, FunSearch is not particularly promising as a technique and the results that it has achieved are not particularly ground-breaking.
- FunSearch, in its current form, can only be applied to a very limited category of mathematical problems. There is essentially zero reason to believe that FunSearch or any related technique will be helpful in solving "a variety of society's pressing scientific and engineering challenges."
- FunSearch uses the LLM as a subroutine in a genetic programming algorithm,[1] to generate mutations of one particular subroutine in a larger program. The LLM is not told either what problem is being addressed or what the overall program is. The prompt consists of two versions of the subroutine, and it is asked to produce another, similar subroutine. The LLM is called millions of times with prompts of this form. Thus, as compared to other application of LLM to mathematical problem solving, the LLM in FunSearch is remarkable for the *shallowness* of the mathematical understanding that it seems to exhibit. Anthropomorphizing, one would say that the LLM is being used for extraordinarily repetitious and tedious work, with no motivation in terms of the ultimate goal that its output serves.

I will first outline the new mathematical discoveries that FunSearch has made (section 2). Then I will explain how FunSearch works (section 3). Then I will discuss the significance of FunSearch (section 4). Sections 2 and 3 are just summaries of the technical content of (Romera-Paredes et al., 2023), highlighting features that

[1]Genetic programming is an AI technique for constructing a good program for some particular problem by starting with a poor program, experimenting with a variety of small changes, analogous to genetic mutations, keeping the new variants that do best on the problem, analogous to adaptive fitness, and iterating.

seem to me important, so readers who have already read that, or who are uninterested in technical details can skip those. All the original material of this paper is in section 4

Necessary disclaimer: I am not an expert in either applications of AI to math or in LLM technology, though I try to keep an eye on both. My knowledge of the mathematical content involved here is extremely limited. I think that my account below is correct, but certainly there is some chance that it has errors of one kind or another; if you notice any, I shall be very obliged if you point them out to me.

2 FunSearch's contributions to mathematics

Romera-Paredes et al. (2023) describe the application of FunSearch to four different mathematical problems:

1. The "cap set" problem, a problem in combinatorics/discrete geometry.
2. "Online bin packing," a problem in combinatorial optimization.
3. "Shannon capacity of cycle graphs," a problem in information theory.
4. The "corners" problem, a problem in combinatorics.

The cap set problem is by far the most extensively discussed, both in (Romera-Paredes et al. 2023) and in the general media accounts of FunSearch. It is certainly the most easily explained and justified for a general readership. My guess, though I cannot be sure, is that the results obtained from FunSearch for cap set were the most interesting mathematically of the four problems and that the scientists who built FunSearch were the most engaged with this problem of the four. In particular, Jordan Ellenberg, a co-author of (Romera-Paredes et al. 2023), is one of the leading experts on the cap set problem and was co-author on one of the major papers in the area (Ellenberg and Gijswijt, 2017, discussed further below).

The online bin packing problem is discussed in the main paper more briefly than the cap set problem, and the other two problems are discussed only in the supplemental material. In this review, I will follow suit, concentrating on the cap set problem and giving only a summary of the results for the other three.

2.1 The Cap Set Problem

The "cap set" problems is unusual among problems of current mathematical research interest in that it can be easily explained to a mathematically interested high-school student.

The problem has to do with the space of n-dimensional vectors whose components are all 0, 1, and 2 where you are doing addition modulo 3. For instance, with $n = 4$, if $\vec{x} = \langle 0, 2, 2, 1 \rangle$ and $\vec{y} = \langle 1, 2, 1, 2 \rangle$, then $\vec{x} + \vec{y} = \langle 1, 1, 0, 0 \rangle$. In ordinary vector geometry, we say that three points $\vec{u}, \vec{v}.\vec{w}$ are collinear if $\vec{w} - \vec{u} = c \cdot (\vec{v} - \vec{u})$ where c is a constant which is neither 0 nor 1. Applying that same definition in the context of arithmetic modulo 3, it must be that $c = 2$, and we can rewrite the condition as "$\vec{u} + \vec{v} + \vec{w} = \vec{0}$". An equivalent condition is that three vectors are collinear if, in each component, either the three vectors all have the same value, or they all have different values. So, for instance, if $\vec{x}$ and $\vec{y}$ are as above, and $\vec{z} = \langle 2, 2, 0, 0 \rangle$ then $\vec{x}, \vec{y}$ and $\vec{z}$ are collinear.

A *cap set* is a set S of vectors such that no three vectors in S are collinear. For instance, for $n = 3$ you can check (with a little work) that the set

0	0	0
0	0	2
0	2	1
0	2	2
1	0	1
2	0	1
2	2	2
2	2	0

is a cap set of size 8.

The *cap set problem* is, given a dimension n, what is the largest cap set of n dimensional vectors, using modulo-3 arithmetic? For people who like this kind of thing, this is a problem of considerable mathematical charm. Moreover, it has deep connections to other combinatorial problems and to the problem of arithmetic sequences in the prime numbers, as well as to the SET card game. (The sequence of values for the maximal cap set is #A090245 in the Online Encyclopedia of Integer Sequences.[2]) There is a very readable account in (Klarreich, 2016) and a mathematically deeper but still comparatively readable account in (Grochow, 2019).

It turns out that the cap set problem is mathematically difficult. Let us use the notation c_n to mean size of the maximal cap set for a particular value of n. No general formula is known for c_n. The exact value of c_n is known for $n = 1 \ldots 6$ but not for $n = 7$ or any larger value. For any particular value of n, you can establish that some value m is a lower bound for c_n by constructing a specific cap set for n of size m. Establishing that m is an upper bound for c_n is a lot harder; you have to show that no cap set for n can be bigger than m.

From the mathematical point of view, more interesting than exact values at particular values of n is the asymptotic behavior; that is, for large values of n, approximately how large is c_n. There is an easy upper bound: since there are only 3^n vectors in total formed out of 0, 1, and 2, certainly $c_n < 3^n$. And there is an easy lower bound; it is easily seen that the set of all vectors with only 0's and 1's but no 2's is a cap set, and there are 2^n of those, so $c_n \geq 2^n$. But that is a large gap. The challenge for mathematicians is to narrow that gap; ideally, to find the exact value $\gamma = \lim_{n\to\infty} c_n^{1/n}$; that is, c_n is approximately equal to γ^n under a suitable definition of "approximately".

Lower bounds for γ can be found by finding a systematic way to generate large cap sets for arbitrarily large n. This can be done by leveraging the idea of an "admissible set of vectors", especially a "full-sized admissible set of vectors". What an "admissible" set of vectors is, and how you can use an admissible set to construct cap sets in arbitrarily high dimension and to calculate a lower bound for γ need not detain us here.[3] All we need to know here is that an admissible set is a finite set of n-dimensional vectors where each group of three satisfies a particular constraint; and that the bigger the admissible set, the more useful it is for constructing big cap sets.

Upper bounds for γ are really tough to establish, since, again, you have to establish that you can't construct a cap set larger than the proposed upper bound. The first proof that there is an upper bound less than 3 here was given by Ellenberg and Gijswijt (2017). They proved an upper bound of $\gamma \leq 2.756$, which remains the best known upper bound.

2.1.1 FunSearch's contribution to the cap set problem

So now we're in a position to discuss what FunSearch contributed to our understanding of the cap set problem.

[2]`https://oeis.org/A090245`

[3]This is, of course, a euphemism for "I have not looked into it, and I am quite sure I wouldn't understand it if I did look into it." The definition of an "admissible" set is given in (Romera-Paredes et al. 2023); for the other two questions, they refer the reader to (Edel, 2004) (I think).

	OR1	OR2	OR3	OR4	Weibull 5k	Weibull 10k	Weibull 100k
First fit	6.42%	6.45%	5.74%	5.23%	4.23%	4.20%	4.00%
Best fit	5.81%	6.06%	5.37%	4.94%	3.98%	3.90%	3.79%
FunSearch	5.30%	4.19%	3.11%	2.47%	0.68%	0.32%	0.03%

Table 1: Comparative results of heuristics, from (Romera-Paredes et al., 2013). The values are the fraction of excess bins as compared to the optimal solution; lower is better. Each column corresponds to a particular benchmark collection of problems.

First: FunSearch found a cap set of size 512 for $n = 8$. The previous best known cap set had size 496. It thus established a lower bound $c_8 \geq 512$

Second: FunSearch found a large admissible set which raised the lower bound for γ from the previous value of 2.2180 to 2.2184.

Third: In examing the output from FunSearch, Ellenberg observed that it correctly assumed a symmetry in the problem of constructing admissible sets that was not previously known. He then was able to use this symmetry to focus the search space that FunSearch was exploring, allowing it to create an even larger lower bound of 2.2202. This feedback loop between the AI and the human mathematician is, to my mind, the most interesting aspect of FunSearch; I will discuss it further in section 4.

2.2 Bin packing

The *bin packing* problem in general is a well known problem in combinatorial optimization: Given a set of bins, all the same size, and a set of objects, none larger than the bins, place the objects into the bins so as to use the smallest possible number of bins. (It is assumed that an object cannot be chopped into pieces which are put in separate bins.) The *online* version of the problem assumes that the objects are presented one by one, and that each object must be placed before the size of remaining objects is known.

The standard algorithms used for the online bin packing problem are first-fit (number the bins and put the object in the first bin where it fits) and best-fit (put the object into the fullest bin where it fits). (I am unable to determine whether these are in fact currently state of the art in terms of overall performance.) The quality of these algorithms is standardly measured in terms of their performance over various benchmark problem collections or collections of problems generated by a specified random distribution. FunSearch found a heuristic that does much better on some collections and significantly better on others than either first-fit or best-fit (table 1).

2.3 Shannon capacity of cycle graphs

Quoting from the supplemental material for (Ramona-Paredes, 2023): "The Shannon capacity of a graph is an important quantity from Information Theory as it indicates the amount of information that can be transmitted over a noisy channel with zero probability of error. Consider a discrete noisy communication channel in which certain symbols can be confused with each other, with a graph describing these confusion patterns. The vertices of the graph correspond to symbols (inputs of the channel), and the edges indicate which symbols can be confused with each other at the receiver side of the communication channel.

One of the most studied graphs with unknown Shannon capacity is the cycle $\mathcal{C}_m$ i.e. the graph with m vertices and m edges forming a single cycle."

A step toward finding the Shannon capacity of cycle graphs is finding large independent sets on particular powers of cycle graphs. In the case of $m = 7$, FunSearch found an independent set that was equal to the biggest known, but simpler and so more susceptible to mathematical inspections. In the cases of $m = 9$ and $m = 11$ it was able to find independent sets larger than had been previously known. Apparently, none of these actually led to an improved estimate of the Shannon capacity for the cycles of size 7, 9, or 11, but they perhaps lead toward one.

2.4 Corners problem

A problem rather similar in flavor to the cap set problem, but less interesting, at least to non-experts like myself. Look it up in the supplementary material, if you're interested. FunSearch allowed the lower bound to be increased from 3.391 to 3.421 in the case $n = 2$ and from 7 to 7.280 in the case $n = 3$.

3 How FunSearch uses the LLM to solve problems

Fundamentally, FunSearch is a genetic programming algorithm which is searching for a program that will produce a good solution to the problem at hand.

It will be easiest to describe the operations of FunSearch for the specific example of the cap set problem; then I will generalize.

In the cap set problem, FunSearch is trying to create a program that will generate a large cap set. The human programmer implements a greedy algorithm `solve(n, priority)`. The arguments to `generateCapSet` are the dimension `n` and a function `priority(v)`, which assigns a numerical priority measure to each vector `v`.

```
function solve(n,priority) {
   L = list of n-dimensional vectors;
   L = sort L in decreasing order of priority(v);
   C = empty set of vectors; % the cap set
   for (v in L) {
      if (v is not collinear with any two vectors in C)
           add v to C;
         }
   return C;
 }
```

The human programmer also programs a function to evaluate the quality of an answer generated by `solve`, in this case just the size of the cap set.

```
function evaluate(solution) {
    return size(solution);
}
```

The FunSearch algorithm is then a genetic programming algorithm. It maintains a population of `priority` functions, scored by the evaluation of the output of the function. It takes two high-scoring priority functions, uses those two as the prompt to the LLM, and implicitly asks the LLM to produce some other priority function of a similar flavor. (Specifically, it identifies the two functions in the prompt as "priority function 1" and "priority function 2" and asks for "priority function 3"). It does this millions of times.

```
function FunSearch(n,solve,evaluate) {
   population = some seed priority functions, generally trivial;
   for i=1:ManyIterations {
      Prompt = choose two high-ranking priority functions from population;
      NewPriority = output of LLM given Prompt;
      if (NewPriority is a valid priority function) {
         NewCapSet = solve(n,NewPriority);
         value = evaluate(NewCapSet);
         add <NewPriority, value> to population
        }
     }
  return (best cap set found);
}
```

The actual FunSearch algorithm is substantially more complicated than the above; it incorporates a number of genetic algorithmic techniques to prevent the system from getting stuck in a poor state, and it is written to support parallel search. However, the above sketch does describe how the LLM is used.

The other applications are analogous. To use FunSearch on a particular problem, the human programmer must supply:

- A *solve* algorithm which takes some key subroutine (in this case `priority`) as an argument, which generates candidate solutions to the mathematical problem. (In general, the mutable function, which is the input and output to the LLM, need not be a priority function, but for the remainder of this paper I will continue to refer to it as a "priority function," since that is a more natural phrase than "mutable function".) Romera-Paredes et al. call this the "skeleton".
- An evaluation function, to measure the quality of any given priority function. For the cap set, Shannon capacity, and corners problem, each priority function generates a single solution to a particular mathematical problem, and the value of the priority function is the value of the solution. For the bin packing problem, the value of the priority function is the average quality of solve applied to problems in a particular benchmark collection.
- Seed versions of the priority function.

That's all. FunSearch and the LLM do the rest.

4 Discussion

I will formulate the discussion in terms of a series of questions. For some of these, I have an opinion as to the answer, of varying degrees of confidence and well-foundedness. For some, I have no guess as to the answer, but the project team knows the answer, and I hope they will publish it. For some, no one knows the answer.

4.1 How impressive is FunSearch as a tool for mathematical research, as compared to previous AI systems?

There have been several applications of AI technology to mathematical research, particularly in the last few years (table 2). (The history of the application of computer technology generally to mathematical research is of course many times larger and more important mathematically.) These vary both in terms of the significance of the mathematical results obtained: the depth and centrality of the AI's contribution to the research enterprise; the generality of mathematical problem to which the AI system potentially might

- 1959. Logic Theorist found shorter proofs of some theorems in *Principia Mathematica.* (Newell, Shaw, and Simon, 1957).
- 1996. The automated theorem prover AQP proved the Robbins conjecture. (McCune, 1997).
- 2014. Hales' proof of the Kepler conjecture was verified using proof assistants Isabelle and HOL Light. (Hales et al. 2017). For further discussion of the contributions of proof assistant technology to mathematical research, see (Avigad, 2024).
- 2021. Deep learning technology is used to suggest conjectures in representation theory and knot theory which led to the proofs of theorems. (Davies et al. 2021; Davis, 2021).
- 2022. Deep reinforcement learning (AlphaTensor) is used to find a number of (in principle) faster matrix multiplication algorithms (Fawzi et al. 2022).

Table 2: Some AI contributions to mathematical research.

apply; the breadth of mathematical problems to the AI system has in fact been successfully applied; and the reasonableness of calling the software "artificial intelligence".

I certainly have no standing to compare the mathematical significance of the results generated by FunSearch to these other results[4] The wide variance of these different accomplishments along the other dimensions makes a meaningful comparison difficult. However, I certainly don't see that there is a clear case to be made that, overall, FunSearch's contribution to mathematics is much greater than that of its predecessors.

4.2 What about the fruitful interaction with Ellenberg?

To my mind, the most interesting aspect of the project was the fruitful interaction between FunSearch and Ellenberg. FunSearch converged on a particular priority function; Ellenberg examined it and realized that it reflected a previously unrecognized symmetry relation in the cap set problem; and then Ellenberg was able to leverage that symmetry to improve the FunSearch search and obtain an even better solution.

Notably, this interaction depended on the fact that FunSearch outputs human-readable code; it could not have happened with an AI that finds its way to large cap sets but whose internal representation is completely opaque, such as a system based on reinforcement deep learning.

Needless to say, 99.99% of the credit for this interaction belongs to Ellenberg rather than FunSearch.

If FunSearch continues to be used, will we see further instances of this kind of fruitful interaction? Impossible to predict. As table 2 suggests, the history of applications of AI to math has seen a number of one-offs, in which an exciting development had no sequels.

4.3 How does FunSearch compare to other applications of LLMs?

As a means of utilizing an LLM for an application, FunSearch is, frankly, off-the-charts weird. Ordinarily, when you use an LLM, you ask the question whose answer you are looking for, and it gives you an answer, based on the material in its training set that is somehow invoked by your question. Perhaps you do a bit of prompt engineering at the start, and perhaps you engage in some further back-and-forth.

[4]In the articles about FunSearch, much has been made of the statement by the distinguished mathematician Terrence Tao (2009) in a blog that the cap set problem was his "favorite" open problem. However, in Tao's blog, the chief question that he hopes to see resolved is whether cap sets of a constant density in the space exist — that is, whether $\lim_{n\to\infty} c_n/3^n > 0$ — which was resolved in the negative in (Ellenberg and and Gijswijt, 2017). How significant Tao or other mathematicians consider FunSearch's findings that $c_8 \geq 512$ and that $\gamma \geq 2.2184$, I can't say.

FunSearch is entirely different. The LLM is never told what problem it is working on or how the output it produces is going to be used. Whatever was in its training set about the cap set problem or even about greedy algorithms is entirely irrelevant. All it is asked, a million times, is "Here are two priority functions; generate some other priority function that would go along with them." Guiding the priority functions toward a solution to the cap set problem is purely the responsibility of the genetic algorithm superroutine, which is invisible to the LLM.

Put it another way: Suppose that we replaced the LLM in FunSearch with an AGI — in fact, suppose we replaced it with an AI that had superhuman mathematical abilities and could directly answer questions like "Please tell me the value of γ and prove the correctness of your answer," or "Please give me a priority function that will produce large cap sets when used in a greedy algorithm." In this archictecture, that AI would have no opportunity to use those abilities, because no one ever bothers to tell it that it is working on a project involving a cap set and a greedy algorithm. There is no reason to think that FunSearch powered by that kind of AI would work at all.

4.4 What kinds of mathematical problems might be promising applications for FunSearch?

The answer given by Romera-Paredes et al. is as follows: "We note that FunSearch currently works best for problems having the following characteristics: a) availability of an efficient evaluator; b) a "rich" scoring feedback quantifying the improvements (as opposed to a binary signal); c) ability to provide a skeleton with an isolated part to be evolved." That seems pretty much right, though I would guess that "works best" should be read "works at all".

Certainly another condition is that the problem is solvable, or approximately solvable, by an algorithm of the specified structure (e.g. greedy algorithm) applied to some attainable priority function. My guess is that it would be rare that one could be sure, or even very confident, of that *a priori.*

Since FunSearch is in effect doing a local (relative to whatever the LLM considers "local") heuristic search through the space of priority functions, I would think another condition would be "existence of a rich space of solutions that are more and less successful".

Certainly, these are more or less necessary conditions for success, but there is no reason to suppose that they are sufficient conditions. That leads directly to the next two questions:

4.5 Why did FunSearch work as well as it did on these four problems?

God only knows, and I doubt that God much cares.

4.6 What if anything does the success of FunSearch indicate?

It certainly indicates nothing about the LLM's knowledge of mathematics or reasoning abilities, since neither of these are called on to any degree.

It would seem to imply something about the topography of the space of priority functions relevant to these particular mathematical problems as mapped by the LLM's idiosyncratic measure of similarity. But it would be very difficult to say anything specific or well-defined about that.

4.7 Is the LLM biased toward short programs or Kolmogorov complexity?

Romera-Paredes et al. propose the following partial answer to my previous question:

Because FunSearch implicitly encourages concise programs, it scales to much larger instances compared to traditional search approaches in structured problems. In a loose sense, FunSearch attempts to find solutions that have low Kolmogorov complexity (which is the length of the shortest computer program that produces a given object as output), while traditional search procedures have a very different inductive bias. We believe that such Kolmogorov-compressed inductive bias is key to FunSearch scaling up to the large instances in our use-cases.

The claim that the LLM is biased toward outputting concise programs *might* be true, and seems somewhat plausible. Presumably the LLM is biased toward producing code that in some respects resembles the code in its training set that is similar to the code in its prompts, however it is that the LLM measures "similarity" and perhaps that creates a bias toward short code. However, in view of our very poor understanding of what LLMs do and in the absence of a careful examination of the bodies of code used in training and fine-tuning the Codey LLM that was used in FunSearch, that is not a very strong argument. As far as I have seen, there is no other evidence to support that claim. Finally, even if the claim is true, it does not go very far toward explaining why FunSearch works well on these problems.

The "loose" usage of "Kolmogorov complexity", on the other hand, is becoming increasingly common in the machine learning community, and is fast becoming a pet peeve of mine. So I will rant about it.

The Kolmogorov complexity of string S in programming language/Turing machine P, denoted "$K_P(S)$", is, as stated above, the length in bits of the smallest inputless program that outputs S. Now, you can write a program that naïvely computes c_n in about 40 lines of Matlab — maybe less, if you're a cleverer programmer than I am — about 2000 bits. The problem, of course, is that it runs in time roughly $3^{n\gamma^n}$ — doubly exponential. On any plausible universal Turing machine, the program may well be considerably longer in bits, but in any case it is pretty short as compared to lots of programs that compute lots of useful things. Let's call the length of that program L. Then we have $K_P(c_n) \leq L + K_P(n)$. If $\log(c_n) > L$ — that is, if $n > L\log(2)/\log(\gamma)$ — then quite possibly this is in fact the shortest program to generate c_n so the above inequality would be tight.[5] The key point here, though, is that this program is not much longer than the code generated by FunSearch — which, keep in mind, includes the greedy algorithm in the skeleton as well as the policy — and it establishes the true value for any n, not just a lower bound for $n = 8$. So if FunSearch had anything to do with Kolmogorov complexity, it would be outputting this program or something similar. It doesn't, and you certainly wouldn't want it to. Kolmogorov complexity is a red herring.

4.8 Is it wise to limit the target code to a single skeleton?

I was quite surprised to see that the FunSearch team had used such a simple-minded algorithm for the skeleton — the simplest possible greedy algorithm, based on priorities which are computed at the start as a function of individual vectors. There are other kinds of priority functions; one could have an evaluation metric over a cap set or a function that maps a cap set to a vector to add, analogous to a policy function in Markov decision processes. And there are many alternative search strategies one could use a beam search, or a stochastic search, or a hill-climbing search with swap operators and sideways motion, and so on. I am not an expert on search, but I would think considering a sophisticated skeleton with a simpler priority function might well do better than the simplest possible skeleton with a comparatively complex priority function. I should certainly think it would be worthwhile experimenting with the option.

Romera-Paredes et al. write,

This [architectural decision] delegates to FunSearch precisely the part that is usually the hardest to come up with. While a fixed skeleton may constrain the space of programs that can be discovered, we find it improves overall results because it focuses the LLM resources on evolving

[5]We may, some time in the future, find out that it is not the shortest program, but we will never be able to prove that it is the shortest program.

> the critical part only, instead of also using the LLM to recreate already known program structures (with more opportunities for mistakes that would render the entire program incorrect).

I agree, certainly, that giving the LLM free rein to invent new skeletons is not likely to be fruitful, but I would say that it's misleading to say that that's because the skeleton "is usually the hardest to come up with". On the contrary, it is because devising a good new skeleton is much harder and error-prone than devising a good new priority function. The fact that the skeleton is protected from the LLM in this way is another indication of how shallow the LLM's understanding of the larger setting is.

4.9 Is FunSearch the best current AI approach to this problem?

Romera-Paredes et al., in their supplementary material, report on some ablation tests and comparative tests on the problem of generating a large admissible set. One result that is intriguing was an experiment where they replaced the LLM by a hand-crafted set of mutations on the priority function, but otherwise left the architecture of FunSearch unchanged. What they found is that the resulting program did a lot worse than FunSearch and took much longer to arrive at anything reasonable, but after 1.5 million iterations had achieved a result that was impressive — better than they expected. Note that this alternative archictecture not only had no LLM; it had no neural-network or deep learning whatever.

There is, however, an enormous difference between FunSearch and this uninformed genetic algorithm in speed of convergence in the early iterations of the algorithm. FunSearch reaches a state that is good, though not optimal, with astonishing speed. In its best run, the seed priority function, with which it begins, gives a score 1161 points below optimal. After 840 calls to the LLM, it reached a score 468 points below optimal. After 88,548 calls, it reached a score 75 points below optimal. Finally after 1,732,100 calls, it reached the optimal. By contrast, the uninformed genetic algorithm had to generate more than a million variants to reach a state 800 points below the optimal. (See figure A.2 of the supplemental information. The exact figures on the best run of FunSearch were kindly provided by Pawan Kumar by email.)

A more important question is this: FunSearch is doing search through a space of complex functions toward a function that maximizes some objective evaluation, using only feedback from the evaluation function on a sequence of candidates. That sounds almost like the definition of reinforcement deep learning. Since this project is coming out of DeepMind, which is the world center for reinforcement deep learning, I couldn't help but wonder: Did they try reinforcement deep learning on any of these problems and if they tried it, how well did it do? I asked Alhussein Fawzi, the corresponding author. who answered:

> We tried initially an RL approach that searches in the space of constructions directly, but that didn't look promising – it didn't scale in particular to large problem sizes. We haven't tried RL in the space of functions, but that'd be an interesting next step.
> (Email from Alhussein Fawzi to Ernest Davis, 12/18/2023; quoted by permission of Dr. Fawzi.)

The article in *Technology Review* (Heaven, 2023) quotes a similar statement:

> Built on top of DeepMind's game-playing AI AlphaZero, both [AlphaTensor and AlphaDev] solved math problems by treating them as if they were puzzles in Go or chess. The trouble is that they are stuck in their lanes, says Bernardino Romera-Paredes, a researcher at the company who worked on both AlphaTensor and FunSearch: "AlphaTensor is great at matrix multiplication, but basically nothing else."

As I remarked above, the history of AI applied to research mathematics has a number of one-offs. The successes of AlphaTensor and AlphaDev suggested that deep reinforcement learning might have a great future in applications to mathematical research; apparently, that tool is much more limited than had been hoped.

4.10 What other mathematical problems, if any, was FunSearch tested on?

To what extent, if any, have the results reported been cherry-picked? The article does not say.

4.11 What was the interpersonal group dynamics of the project?

This is not a question that one usually asks about a technical paper, but since the product here is being touted as potentially a useful tool for the working mathematician, it seems to me legitimate. We have here a team from DeepMind plus a mathematician who is a world expert on the cap set problem, and we have the report of the outcome of a new software system, primarily on cap set and secondarily on three other problems. How did this come about? Did Ellenberg propose the cap set problem to the Deep Mind team, and they developed FunSearch to solve that problem? Was FunSearch developed independently, and then someone at DeepMind thought to propose it to Ellenberg? Looking forward to future collaborations between mathematicians and AI labs, this would be useful to know.

4.12 What ultimately will be the impact of FunSearch?

Obviously, there is no way to predict with certainty. But in view of its clear limitations and its strange construction, my own feeling is that, among the many different techniques that have been proposed and attempted for applying AI to mathematics — symbolic proof construction and proof verification, proof construction using learned strategies, using LLMs to translate mathematics into code, using LLMs to translate mathematics into formal logic, applying deep reinforcement learning — FunSearch seems like one of the most niche and least promising. Certainly claims that this particular technology is revolutionary or even a great leap forward seem unfounded.

Acknowledgements

Thanks to Gary Marcus and Andrew Sundstrom for helpful feedback on this review.

References

J. Avigad (2024). "Mathematics and the formal turn". *Bulletin of the American Mathematical Society,* to appear. `https://arxiv.org/abs/2311.00007`

D. Castelvecchi (2023). "DeepMind AI outdoes human mathematicians on unsolved problems." *Nature* news item.
`https://www.nature.com/articles/d41586-023-04043-w#ref-CR1`

A. Davies et al. (2021) "Advancing mathematics by guiding human intuition with AI," *Nature,* **600**:70-74. `https://www.nature.com/articles/s41586-021-04086-x`

E. Davis (2021). "Deep Learning and Mathematical Intuition: A Review of (Davies et al. 2021)." arXiv preprint 2112.04324. `https://arxiv.org/abs/2112.04324`

Y. Edel (2004). "Extensions of generalized product caps." *Designs, Codes, and Cryptography,* **31**:5-14.

J. Ellenberg and D. Gijswijt (2017). "On large subsets of F_n^q with no three-term arithmetic progression, *Annals of Mathematics,* 339-343.

A. Fawzi (2022). "Discovering faster matrix multiplication algorithms with reinforcement learning." *Nature,* **610** (7930):47-53. `https://doi.org/10.1038/s41586-022-05172-4`

A. Fawzi and B. Romera-Paredes (2023). "FunSearch: Making new discoveries in mathematical sciences using Large Language Models". Google DeepMind Blog.
`https://deepmind.google/discover/blog/funsearch-making-new-discoveries-in-mathematical-sciences-using-l`

J. Grochow (2019). "New applications of the polynomial method: the cap set conjecture and beyond." *Bulletin of the American Mathematical Society,* **56**(1), pp.29-64.

T. Hales et al. (2017) "A formal proof of the Kepler conjecture." *Forum of mathematics, Pi*, **5**, p. e2.

W.D. Heaven (2023). "Google DeepMind used a large language model to solve an unsolvable math problem." *Technology Review.*
`https://www.technologyreview.com/2023/12/14/1085318/google-deepmind-large-language-model-solve-unsolvabl`

E. Klarreich (2016). "Simple Set Game Proof Stuns Mathematicians." *Quanta Magazine,* May 31, 2016.
`https://www.quantamagazine.org/set-proof-stuns-mathematicians-20160531/`

W. McCune (1997). "Solution of the Robbins problem." *Journal of Automated Reasoning,* **19**:263-276

A. Newell, J.C. Shaw, and H.A. Simon (1957). "Empirical Explorations of the Logic theory Machine: A Case Study". *1957 Western Computer Proceedings,* 218-230. `https://dl.acm.org/doi/pdf/10.1145/1455567.1455605`

B. Romera-Paredes et al. (2023). "Mathematical discoveries from program search with large language models". *Nature* accelerated article preview.
`https://doi.org/10.1038/s41586-023-06924-6`

T. Tao, (2009). "Open question: best bounds for cap sets."
`https://terrytao.wordpress.com/2007/02/23/open-question-best-bounds-for-cap-sets/`