AXPLORER – CONSTRUCTION D'EXEMPLES UTILISANT DES METHODES D'IA GENERATIVE

FRANÇOIS CHARTON, AXIOM, ENPC

FACETS OF AI FOR MATHS

- LLM-assisted discovery: literature search on steroids
  - A few spectacular results (Erdos)
  - A lot of crowdsourced noise
  - LLM have a strong bias towards past research
- Theorem proving
  - AI can assist Lean provers
  - Scaling requires autoformalization: caveat emptor!
- Bespoke AI as a tool for mathematical discovery
  - Models trained from scratch, on specific problems
  - To solve a hard problem (Lyapunov functions, Alfarano, C, Hayat 2024)
  - Provide new insight (Collatz, C. Narayanan, 2025)
  - Build interesting mathematical objects

GENERATIVE AI

- Train a model to generate particular sentences, or images
  - By learning the next letter in a word, the next word of a sentence, the next pixels in an image
  - By inverting the diffusion process that blurs an image
- Learn a conditional distribution of the "next token", or some statistical method for generation
- Use it to generate more
  - Makemore: children (or brand) names
  - GPT: answers from questions
  - Sora: image from prompt

PATTERNBOOST – GENERATIVE AI FOR COMBINATORICS (FC, Ellenberg, Wagner, Williamson, 2024, 2411:00566)

- To solve a combinatorial problem (find the best element in some class)
- Train a next-token prediction model on good candidate solutions
- Use the train model to generate new candidates, hopefully better
- Feed the best solutions back into the model, to generate even better solutions

PATTERNBOOST – A RECIPE

- Generative method for discrete optimization

- You want to solve a discrete optimization problem
  - Special elements of a class, the interest of which can be measured by a reward function
  - Largest graph / grid / set verifying some constraint

- You know how to generate half-decent (or random) candidates
- You can measure the quality of a candidate
- You have a ”local search” that can correct a candidate, and improve it
  - It does not need to be good
  - But it should be fast

SQUARE FREE GRAPHS

- What is the largest undirected graph over N nodes, with no 4-cycle
  - No 4 distinct nodes A,B,C and D such that A—B, B—C, C—D and D—A

- Generating random graphs
  - Start with an empty graph
  - Add a random edge, reject if it creates a cycle
  - Stop after K unsuccessful attempts
- Measuring quality : just count the edges
- Correcting / improving a graph:
  - Start with a graph with 4-cycles
  - Cut one edge off every 4-cycle until none are left
  - Try to add random edges that do not create cycles (stop after K attempts)

GENERATING CANDIDATES WITH AI

- Generate an initial set of candidates
  - Many random solutions, keep the best
  - Represented as "sequences of words" (a sequences of edges, an adjacency matrix)
- Use them to train a language model
  - A small decoder-only transformer
  - Trained from scratch to predict the next "word" in the solution
- Use the model to generate new candidates
  - Leveraging the learned next word probability
  - From a random first word, predict a "likely solution" (according to the model)
- These candidates can be incorrect solutions: use the local search to redeem and improve them
- Keep the best candidates, and feed them back into the model
- Rinse, repeat, and hope for the best…

WHY WOULD THAT WORK?

- All happy families are alike; each unhappy family is unhappy in its own way (Tolstoi)

- Good candidates share common traits (whereas bad ones are all over the search space)
  - These common traits can be learned by the generative model
  - And used to generate better candidates
- A "guided search" algorithm
  - Population-based: reminiscent of genetic algorithms

PATTERNBOOST SUCCESSES

- Could discover new solutions to open problems
  - Solved a 30 years old conjecture

- Was used successfully by other teams
  - Sun, Ramos: new counter-examples to an Erdos conjecture about the log-concavity of independence sequences of graphs
  - Jacobi, Williamson: construction of Hadamard matrices
  - Hashemi, Berczi: FlowBoost, flow-matching meets PatternBoost, sphere packing

PATTERNBOOST FAILURES

- Fails on hard cases
  - Slow, needed a lot of compute
  - And a good initial sample
- Alternative tools were introduced, Alpha/Open/Shinka/Theta Evolve, using Code Generation LLM
  - Instead of constructing mathematical object, write programs that construct them
- Should we still use PatternBoost?
- Can we scale it?

THE NO 4-CYCLE PROBLEM

- What is the largest undirected graph over N nodes, with no 4-cycle
  - 4 distinct nodes A,B,C and D with A—B, B—C, C—D and D—A
- A well-studied problem
  - Introduced by Erdos in 1938
  - A theoretical bound $C_n \leq \frac{n}{4}\left(1+\sqrt{4n-3}\right)$ was proposed by Reiman (1958), not sharp but asymptotically correct (Erdos, Renyi, Sosm 1966)
  - Sharp bounds are known in particular cases
    - $C_{1+q+q^2} = \frac{q}{2}(q+1)^2$ for q powers of primes, and $C_{1+q+q^2} \leq \frac{q}{2}(q+1)^2$ in the general case (Füredi, 1983, 1996)
    - $C_{q+q^2} \leq \frac{q}{2}(q+1)^2 - q$ for even q (Firke, Kosek, Nash, Williford, 2013)

COMPUTATIONAL STATE OF THE ART FOR THE 4-CYCLE PROBLEM

- 1989: solutions have been computed for all n up to 21 (Clapham, Flockhart, Sheehan)
- 1992: all n up to n=31 (Yang, Rowlinson)
- 2025: all n up to n=40, competitive solutions for n up to 49
  - Brendan McKay's page https://users.cecs.anu.edu.au/~bdm/data/extremal.html

TOO HARD FOR PATTERNBOOST?

- Could solve the problem for all n up to 30
  - For n=30, we generated 78.3 million candidates
  - And needed to run a dozen experiments for one to succeed
- And n=33, but…
  - We ran over 100 models, for over three weeks,
  - The one successful model generated 116.5 million candidates
  - Better than brute force search, but far from a practical solution

n	Best	Best found	#Searches	n	Best	Best found	#Searches
20	46	**46**	0.4	38	117	110	77.1
21	50	**50**	0.8	39	122	113	11.7
22	52	**52**	0.4	40	127	117	11.2
23	56	**56**	2.3	41	$\geq$132	102	8.6
24	59	**59**	1.8	42	$\geq$137	124	6.8
25	63	**63**	3.1	43	$\geq$142	128	15.6
26	67	**67**	8.4	44	$\geq$148	132	12.1
27	71	**71**	21.0	45	$\geq$154	136	11.1
28	76	**76**	23.8	46	$\geq$157	140	10.4
29	80	**80**	41.7	47	$\geq$163	144	12.1
30	85	**85**	78.3	48	$\geq$168	147	1.9
31	90	87	126.3	49	$\geq$174	152	12.2
32	92	91	104.9	50	?	156	15.2
33	96	**96**	116.5	51	?	161	14.7
34	102	95	39.9	52	?	165	14.1
35	106	100	121.3	53	?	169	5.2
36	110	102	38.4	54	?	173	9.0
37	113	106	29.8	55	?	177	8.2

AXPLORER: A STREAMLINED VERSION

- All Python, optimized models, simpler to use and extend
- Uses less memory, small compute friendly
- Runs on MacBook Pro (mps), CUDA, and potentially others
- Open source https://github.com/AxiomMath/Axplorer

THE AXPLORER CONFIGURATION

- Start with 100,000 candidate solutions
- Train a model for 50,000 optimization steps, with batches of 32
- Use the model to generate 500,000 candidates
  - Not all of them square-free
- Redeem / optimize them via a basic search algorithm
  - Find all squares in the graph
  - Remove them, focusing on edges shared by several squares (if any)
  - Greedily add random edges, until we cannot
- Keep 200,000 best, and train the model
- Repeat…

AXPLORER AT WORK

- For n=33, the hardest problem solved by PB (100 GPU, 3 weeks, hundreds of epochs), Axplorer finds a solution in 5 epochs
  - 2.6 million generated candidates (vs 116 in PB)
  - 2.5 hours on a L4 GPU (3$ on a cloud provider)
  - All experiments succeed in less than 8 epochs
- As training proceeds, the distribution of generated examples shifts towards larger (and better) candidates, with no loss in diversity

Edges	Initial	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
76	294	14				
77	1415	313	2			
78	4745	3214	70	1		
79	11933	18053	880	28	1	
80	20571	58938	7827	652	31	4
81	**22978**	116198	35637	5753	531	77
82	16888	**137091**	89904	28105	4720	969
83	8011	99086	**131778**	80724	24524	7234
84	2493	44102	112431	**134056**	72756	32608
85	523	12465	56817	129488	127593	83564
86	75	2298	17300	73263	**129200**	**126345**
87	9	266	3414	25800	78807	116891
88		22	425	5561	30341	68822
89		2	31	775	7735	27006
90			3	68	1447	7910
91			1	9	222	1927
92					43	430
93					8	99
94						39
95						6
96						**1**

AXPLORER AT WORK

- Best solutions could be recovered for all n up to 40 (in less than a day)
  - N=30: 2.1M examples
  - N=33: 2.6M
  - N=34: 3.6M
  - N=36: 6.6M
  - N=37: 10.1M
  - N=38: 12.1M
  - N=39: 12.5M
  - N=40: 13.1M
- Competitive solutions up to N=45

HOW IS IT DONE?
LEARNING I: ENCODING THE GRAPH

- A graph is a triangular binary matrix, with N(N-1)/2 entries
- Our candidates are sparse

- 2 main encoding techniques
  - Dense: a sequence of zeroes and ones
    - Small vocabulary, long sequence
    - Can be made shorter by "grouping" entries
      - E.g. in blocks of 4 encoded 0 to 15
      - Byte Pair Encoding, used in PB1
  - Sparse: a sequence of non-zero entries
    - Shorter sequence, larger vocabulary
    - Graph = list of edges

LEARNING I: ENCODING THE GRAPH

- In a transformer
  - Large vocabularies are processed by the embedding (a simple linear layer)
  - Long sequences are processed by the attention mechanism (complex and quadratic), more powerful but slower
  - You want to **trade these off**
- Sparse encoding are clearly beneficial
- N=30,
  - Sparse encodings: find an optimal solution (85 edges) in 4 epochs
  - Dense encodings: find graphs with 79-81 edges after 30 epochs
    - And are much slower because the sequences are longer

SPARSE AND SPARSER

- Edges can be encoded as single words,
  - A graph with E edges over N nodes is a sequence of E tokens, over a vocabulary of N(N-1)/2 words
- Or pairs of nodes
  - The graph is a sequence of 2E words over a vocabulary of N

- For n=38 (best solution: 117), over 4 experiments
  - Edge encoding recovers solution in 28-42 epochs
  - Pair-of-node encoding recovers solution in 18-29 epochs
    - But run slower, because the sequences are longer

LEARNING II: MODEL SIZE MATTERS LESS

- Base experiments use a 4-layer transformer, with dimension d=128
  - Model size grows linearly in l, but quadratically in d
- N=38,
  - 4/128 model, recovers solution in 18-29 epochs
  - 4/256 model in16-19
  - 6/128 model in 19-27
  - 6/256 perform less well,
    - much slower
    - need more data when trained from scratch
- There is a small gain in having larger or deeper models, but no visible scaling laws
  - much larger models?

LEARNING III: TEMPERATURE MATTERS

- New candidates are generated by sampling the next token probability learned by the model
- Before sampling, these values are normalized using a softmax operator $p_i = \frac{e^{\frac{q_i}{T}}}{\sum_j e^{\frac{q_j}{T}}}$
  - T is the "temperature", which trades off predicted performance for diversity
  - A large temperature will generate a broader set of candidates (and care less for predicted performance)
  - A low temperature will generate less diverse candidates (that the model predicts to be good)
    - At very low temperatures, we generate many duplicates

LEARNING III: LISTEN TO THE MODEL!

- As a rule, a low temperature helps:
- For N=29
  - T=0.9 never recovers
  - T=0.8 recovers in 17 epochs
  - T=0.7 recovers in 13 epochs
  - T=0.6 recovers in 5 epochs
  - But T=0.5 and 0.4 perform less well (10 epochs, and no recovery in 20)

- A low temperature boosts the model initial progress
- But causes the model to generate duplicates, and fail to improve further

LEARNING III: ANNEALING

- The number of duplicates in the generated sample measures the adverse effect of low temperature
  - As training proceeds, it grows exponentially
- Increase temperature as soon as a small percentage of duplicates appear

LESSON IV: IS LOCAL SEARCH NECESSARY?

- Local search plays two roles
  - Redeems incorrect generated examples: graphs with 4-cycles
  - Improves generated examples
- Redemption is necessary: even late during training, most generated examples have cycles
  - The model **does not learn** to generate square free graphs
  - It generates graphs **that can be corrected** to large square free graphs
- Local search is responsible for discovery
  - Without it learning progresses very slowly, and stagnates

- The generative model explores
- Local search exploits

FAILURE MODE: DIVERSITY COLLAPSE

- After many generations, all examples have the same number of edges, and the model stops learning
  - Copies of the same graph (up to a reordering of its nodes)

- No obvious solution
- Deduplicating, e.g. by sorting nodes according to their number of neighbors, degrades results

CONCLUSIONS AND FUTURE DIRECTIONS

- A low hanging fruit : PatternBoost proved easy to improve
  - Try it! https://Github.com/AxiomMath/Axplorer
- Many paths for improvement
  - Any generative method could work
    - Diffusion Models (FlowBoost, sphere packing problem, Hashemi Berczi)
    - Techniques from genetic algorithms
  - Extend to a larger class of problems: learning a conditional probability:
    - Learning special triangulations of polytopes to help classify toric Calabi Yao manifolds (Yip, Arnal, C, Shiu)
    - Encoder-decoder architectures
  - Other reinforcement learning approaches could be used (instead of just finetuning on new candidates)