Exact Algorithms for Recognizing Sumsets:

Using Concepts from Additive Combinatorics to Identify Structure in Sets

Question:

"Can we reconstruct a set given the aftermath of an operation?"

Sumset Recognition:

Given a set X containing natural numbers

$$X = [\, x_1 , x_2 , x_3 , \ldots , x_{n-2} , x_{n-1} , x_n \,]$$

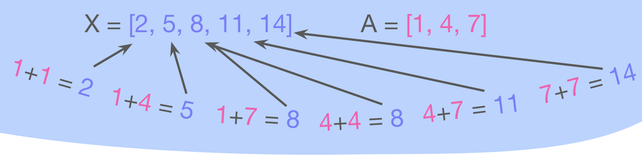
Find a set of natural numbers A:

$$A = [\, a_1 , a_2 , a_3 , \ldots , a_{m-2} , a_{m-1} , a_m \,]$$

satisfying $A + A = X$.

$A + A := \{a + b \mid a, b \in A\}$

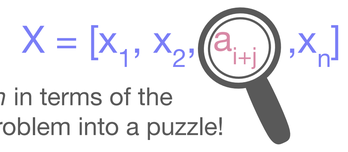
Example:



Why Study Exact Algorithms?

- Brute-force algorithms enumerate *all* possibilities, and find a solution in that long list.
- An **exact** algorithm can solve all instances of an NP-complete problem *faster* than the brute-force enumeration.
- Brute-force could technically be used to solve all NP-complete problems, but it is extremely slow, and not very clever, so we're looking for interesting mathematical structure or facts that could give us a shortcut to the solution.

Set Reconstruction, Broadly



- By thinking about *Sumset Recognition* in terms of the reconstruction of a set, we turn this problem into a puzzle!
- Now, our job is to find the clues for A that already exist in X.

Methods

- From our initial dive into surrounding literature, we found the *Sumset Recognition* problem hadn't been widely explored from an algorithmic perspective, though it was proven to be NP-complete (Abboud, Amir, et al.). So, we would need to chart a whole new territory in order to make progress!
- To organize our thinking, we thought about the problem space through the lens of two approaches: **top-down** and **bottom-up**.

Structural Observations and Building An Intuition

Assumptions: When discussing the form of X and A, we assume them to be ordered (least to greatest) and all elements within them to be unique integers.

Observation 1 (Starting Values): If A+A = X, then x_1, x_2, x_{n-1}, and x_n must take the form:

$$X = [\underbrace{a_1 + a_1}_{x_1}, \underbrace{a_1 + a_2}_{x_2}, \ldots, \underbrace{a_{m-1} + a_m}_{x_{n-1}}, \underbrace{a_m + a_m}_{x_n}]$$

This lets us find a_1, a_2, a_{m-1}, and a_m in constant time!

Observation 2 (Internal Orderings in X):
We know pairwise sums of A must exist within X in the following orders:

(a) $a_i + a_1 < a_i + a_2 < a_i + a_3 < \ldots < a_i + a_{m-1} < a_i + a_m \,, (\forall a_i \in A)$

(b) $2a_1 < 2a_2 < 2a_2 < \ldots < 2a_{m-1} < 2a_m$

(c) $\text{If } i \le j < k \le l \implies a_i + a_j < a_k + a_l$

Definition 1 (Candidate Sets):
We call any set C such that it is guaranteed $A \subseteq C$, a candidate set.

Ex: $A \subseteq X - a_i$ (from observation 2a)
$A \subseteq X/2$ (from observation 2b)

TOP DOWN

Top-down: Worst-Case Improvements

The top-down approach looks at the problem at its most broad: considering the worst-case, with no additional restrictions. While it can seem expansive, we've made progress while looking at the problem through this lens, developing a brute-force branching algorithm as well as a graphical solution. Both of these approaches depend on having a candidate set, thus to reduce runtime it is important to have the smallest set possible with respect to |X| = n.

Candidate of Size n/2

The tightest candidate set we've found is formed as follows:

$$C_{goat} := (X / 2) \cap (X - a_1) \cap (X - a_m)$$

From observations 2a and 2b it is clear that this set must superset A and is therefore a candidate set.

Brief Explanation of |C| < n/2

Note that the elements from X that fall in the range $[a_1, a_m]$ after subtracting a_1 and a_m must fall before $a_1 + a_m$ and after $a_1 + a_m$ respectively (via observation 2a). Thus each candidate must have a representative in the "left" and "right" side of set X, meaning it is of size at most n/2.

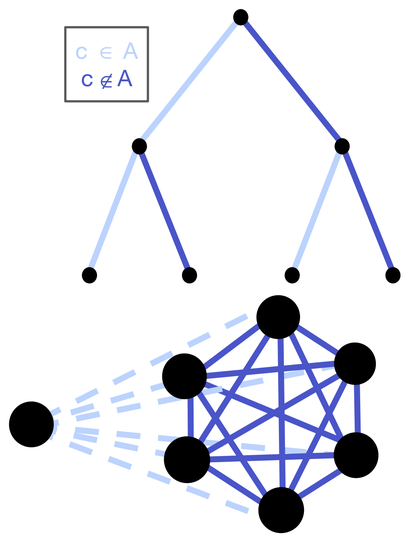
Brute Branching Algorithm

1. Form C_{goat}.
2. Test every subset of the candidate set to see if they produce X.
3. Return any successful subsets A.

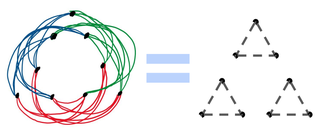
The runtime of this algorithm is $O^*(2^{|C|})$, as we can choose to include or exclude each candidate from our set A. Thus, using the result for our tightest candidate set, the worst-case run time becomes $O^*(2^{n/2})$.

Clique Finding Algorithm

1. Form the candidate set as described
2. Turn every element in the candidate set into a vertex
3. Create an edge between vertices if the sum is in X.
4. Find all maximal cliques.
5. Return a maximal clique if those candidates form X.



Maximal Clique Worst Case



For our clique-finding algorithm to solve the problem in the most general case, our worst-case ends up being a graph structure colloquially called the **"anti-triangle"** structure. In this structure, vertices are sectioned into groups of three where they are connected to every other vertex in the structure *except* the vertices in that group.

The anti-triangle structure leads to a worst case bound of $O^*(3^{n/3})$ because it maximizes the amount of maximal cliques. Since we also cut the candidate set in half, we get a worst case bound of $O^*(3^{n/6})$.

Next Steps

Our next steps from the *top-down* perspective are to further investigate the equivalencies that are needed to make the triangle case possible. Our intuition is that it is impossible to occur, due to there needing to be strict limitations and side possibilities to form it. We are now currently trying to define a procedure to consistently generate the **anti-triangle** case, to create one or fully disprove its existence.

BOTTOM UP

Bottom-up: Perfect and Average-Free Cases

The bottom-up approach involves defining subproblems that we can solve efficiently. That is, we want to narrow the problem into smaller parts that are easier to solve. We can impose more structure on these smaller parts, which often allows us to make conjectures!

One of the main questions we asked ourselves with this approach was:

"Fixing some property of X, can we find A more efficiently?"

And to answer this, we identified two types of set structures: ***Perfect*** *and* ***Average-Free.***

Perfect Sumsets

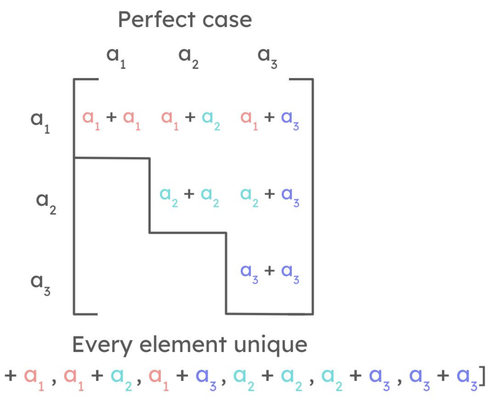
A sumset X is **perfect** if there exists A such that

$$A + A = X,$$

where all pairwise sums in A are unique.

A = [1, 3, 6] X = [2, 4, 6, 7, 9, 12] VS A = [1, 5, 9] X = [2, 6, 10, 14, 18]

(5 + 5 = 1 + 9 = 10)



An Efficient Algorithm for Perfect Sumsets

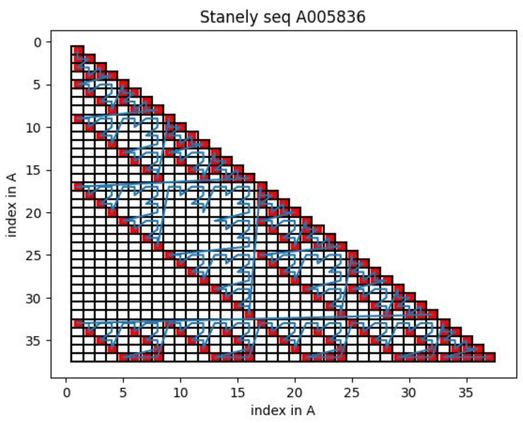
1. We know $a_1 + a_n$ will be the lowest element in X, so subtract a_1
2. Add a_n to A
3. Add a_n to each element in A, and subtract the sum from X
4. Repeat this process until X is empty, then we found A

Average-free Sumsets

Average-free sumsets are sumsets which lack any arithmetic progression (AP) of three numbers: they don't contain any subsets like [1, 2, 3] or [4, 8, 12]. For that reason, they're also known as 3-AP-free sets. The central claim surrounding average-free sets has to do with the size of their A relative to their sumset A + A. We believe that if we know a solution set A is average-free, we also know that the size of A + A must be = ω(size of A).

Next Steps

Our next steps from the *bottom-up* perspective are to discover more about the structure of **average-free** sumsets and try to create a more efficient algorithm for them than our general case algorithm. We believe that due to the extra structure, we will be able to find time improvements that wouldn't have been possible otherwise.



The above grid represents the additions between elements of A. The blue line illustrates the ordering of the elements of X. If two additions of A are equivalent, the representative closer to the diagonal is chosen along the path.
Red indicates a member of X that is unique, i.e., there exists only one sum in A that equals this element.

Contributions and Thanks:

Team Members:
Professor Tim Randolph
Theo Julien
Riley Brown
Mia Alexander

Acknowledgements:
We would like to thank everyone who played a role in making this research possible: Professor Tim Randolph, Chelsey Calingo, Kevin Herrera, Nic Dodds, Landon Tu, Selene Ye, and Bill Lenhart.

Funding:
This project was funded in part by contributions from the National Science Foundation under Grant #2243941, and in part by Harvey Mudd College's Computer Science Department.