Bin Packing to Max 3-SAT Reduction

Cullen Gostel

Why 3-SAT?

- Given our options, we need to reduce an algebraic and combinatorial problem into either a graph, set, or satisfiability problem.
- Graphs are fundamentally about pairwise relationships, don't lend themselves well to reducing higher order problems.
- A set problem can encode the original problem cleanly, but reductions take exponential time.
- Satisfiability can constrain both the algebraic and combinatorial components cleanly. I'll elaborate on this now.

Reduction Approach

- Variables: For each item "u" and each bin "j", we create a variable "x_u_j" that is
is in bin j, and false if it is not.
- We need to constrain the placement of items – each item needs to be placed ir
1 bin. We ensure each item is in a bin by creating a long OR for each item (x_u_0 v x_u_1 v x_u_2 v ...). To make sure they aren't in 2 bins: for every pair
and item, add a clause (-x_u_j1 v –x_u_j2).
- We enforce the capacity constraint by constructing adder circuits using AND/C
gates, essentially using the 3-SAT solver as a calculator.
- How do you convert adder circuits to 3-SAT? Using the Tseitin transformation,
takes an arbitrary combinatorial circuit and converts it to a SAT instance.

Why this works

- If the 3-SAT solver chooses an invalid instance of variables (i.e., an instance w
size is unassigned to a bin, assigned to 2 bins, or exceeds capacity) - then one
clauses will be violated and 3-SAT will be forced to backtrack.
- Using the adder circuits, the sum of the sizes in each bin is compared to the c
using a logical comparator.
- There is only a solution to the Bin Packing problem if the SAT instance is fully
satisfiable. Any clause not being satisfied indicates we violated the logical con
(items in no or >1 bin) or the algebraic constraint (capacity).

Note

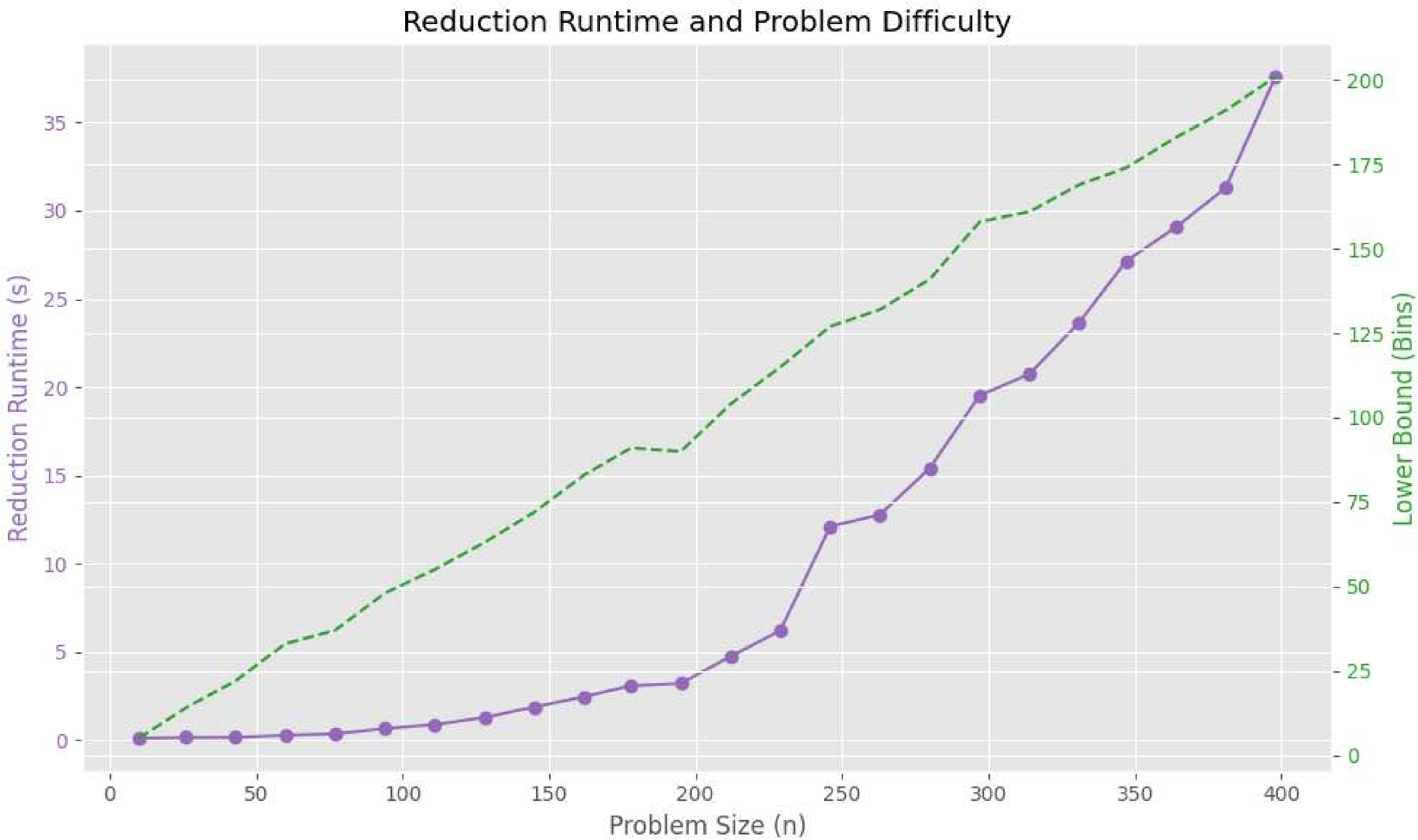
- This reduction works on the NP-Complete, decision version of bin packing: i.e. these n items fit in k amount of bins," as opposed to the NP-Hard "what is the amount of bins to store n items?"
- Additionally, we are only interested if the clauses are fully satisfiable – the "ma satisfiability is irrelevant if it is less than the number of clauses.
- Can solve the NP-Hard version by picking an arbitrary k to start with, and then searching for the minimum k.

Bounds Calculation and Runtime

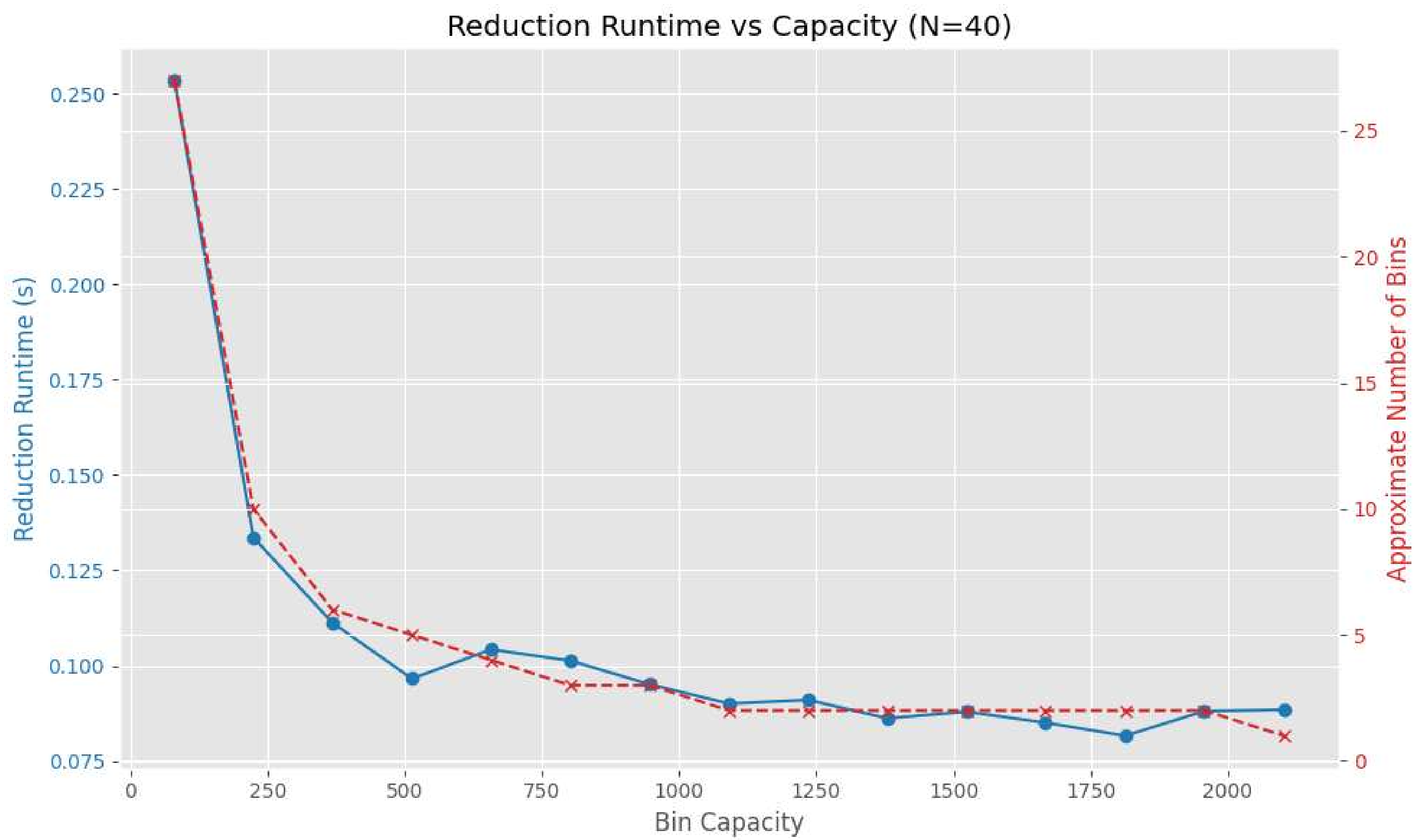
- The lower bound of the optimal value is (the sum of the sizes) / the capacity
- The upper bound is the number of sizes (each size maps to 1 bin)
- The lower bound of the clauses generated is the number of sizes (this is the c where each item can only fit in 1 bin)
- The upper bound of the clauses generated (effectively the Big-O runtime) is

$O(n * m^2 + n * m * L)$ where n is the number of sizes, m is the number of bins the number of bits needed to represent the capacity (log capacity)

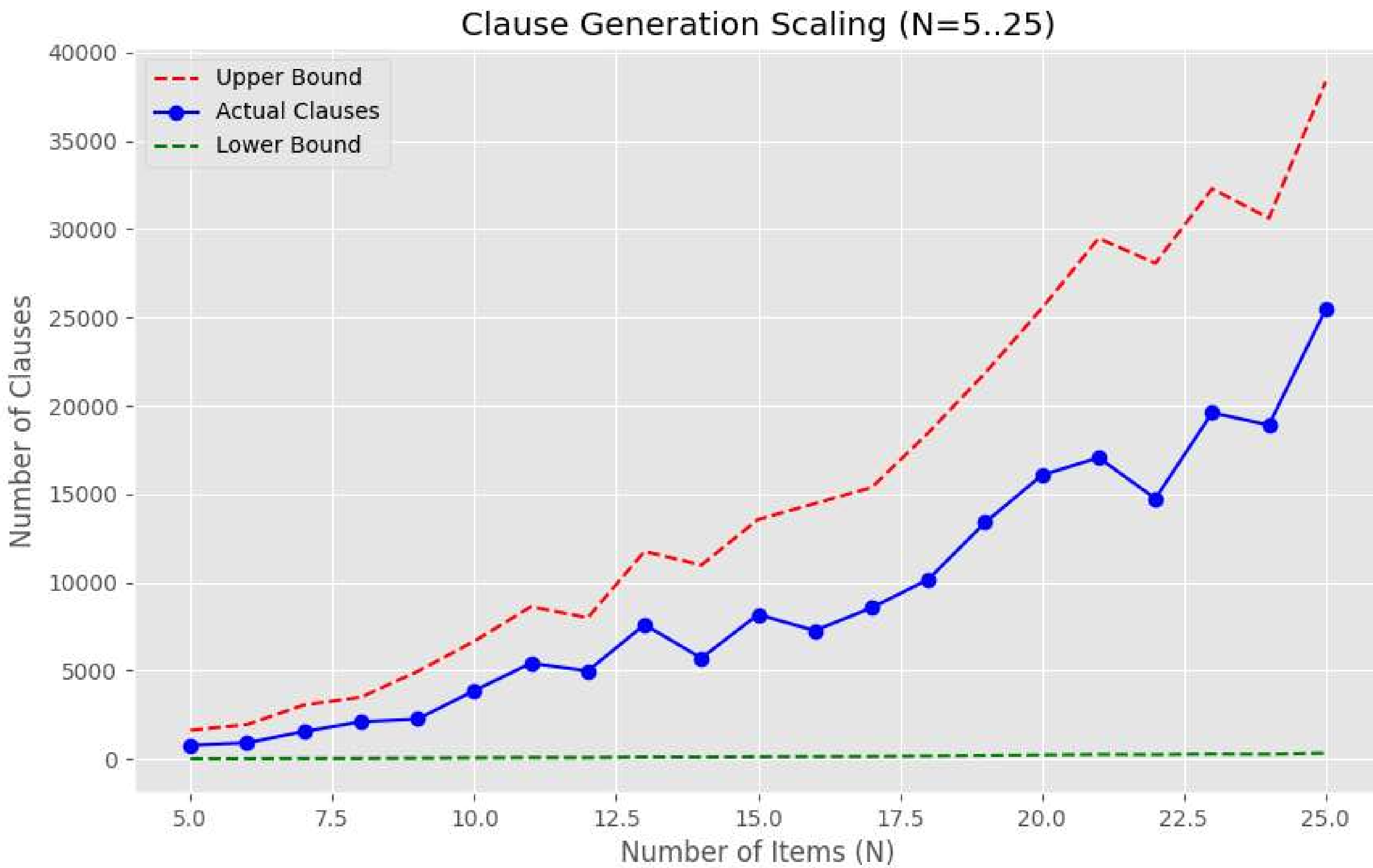
Reduction Runtime vs Problem Size



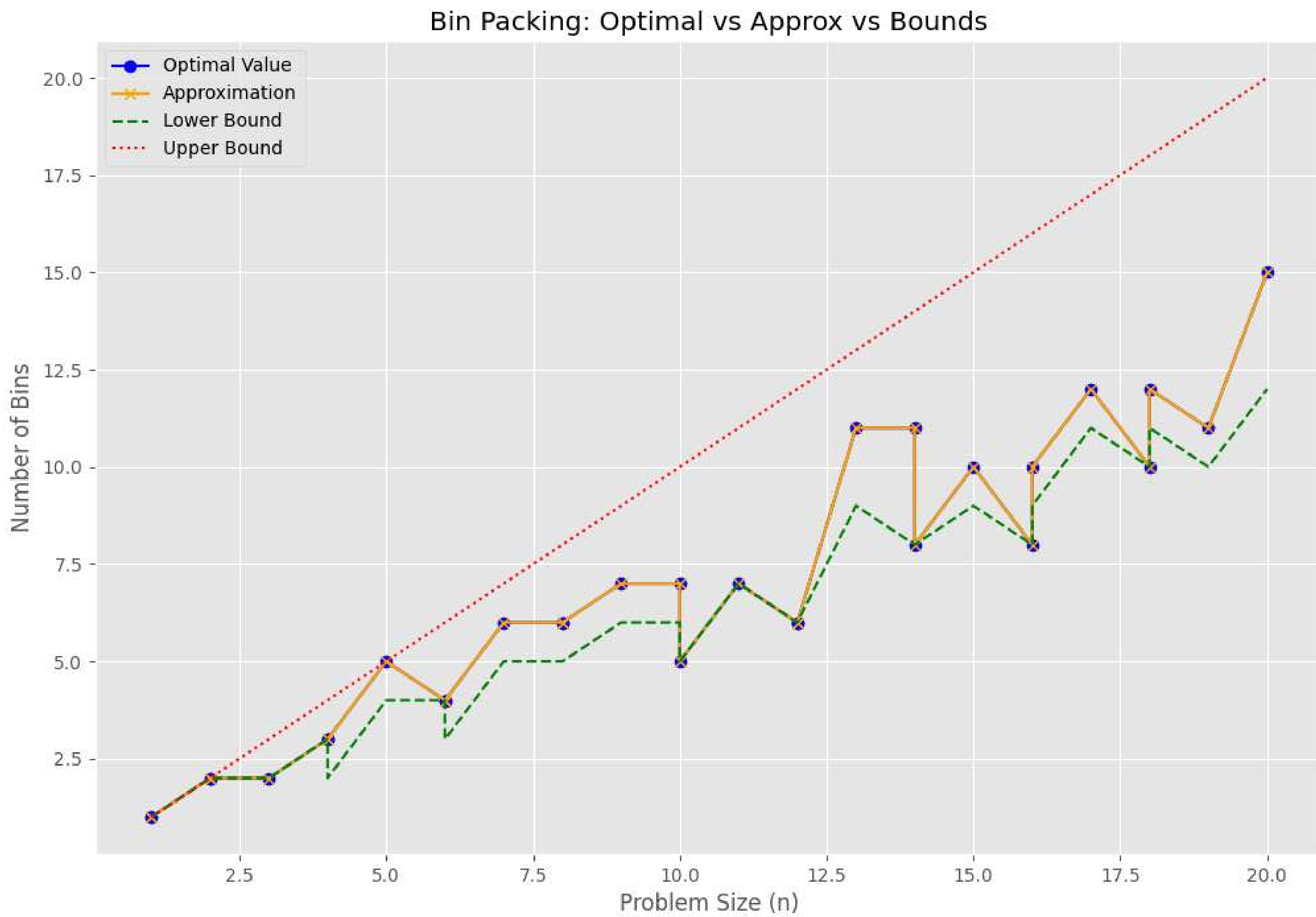
Reduction Runtime vs Capacity



Input Size vs Number of Clauses



Optimal vs Bounds and Approximate vs Bo



Optimal vs Bounds and Approximate vs Bo