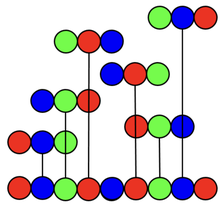
Toward Finding the Minimal Superpermutation on Six Symbols

Alex Knox and Kyle Booker Advised by Marijn Heule
{akknox, kbooker}@andrew.cmu.edu
Carnegie Mellon University

Introduction

A **superpermutation** on n symbols is a string that contains each permutation of n symbols as a substring.



The **minimal superpermutation** problems is: For some n, what is the minimum L, such that there is a superpermutation of length L?

For $n \leq 5$, minimal bounds match values predicted by a constructive algorithm, but the bounds for $n = 6$ do not.

n	Construction	Min L
2	3	3
3	9	9
4	33	33
5	153	153
6	873	$867 \leq L \leq 872$

Methodology

SAT solvers are automated reasoning tools designed to efficiently solve problems in propositional logic.

By designing novel encodings of the minimal superpermutation problem in propositional logic, how can we **use SAT solvers to *verify* and *improve* the known bounds** for minimal superpermutations on $n \geq 6$ symbols?

Encodings

Naive Encoding

We interpret a superpermutation as an array of symbols that contains every permutation at least once.

Basic Cost Model

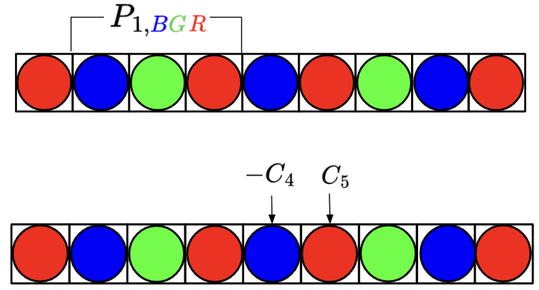
Knowing that $n!$ permutations must be completed in L symbols, we say that additional symbols incur cost if they do not complete a permutation.

- No permutation is repeated
- The total cost is bound by C: $C = L - n! - (n-1)$

Higher Order Cost Model

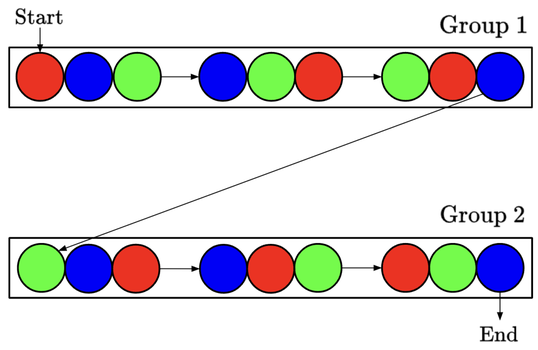
Leaving every group incurs at least one unit of cost in the Basic Cost Model, and we know every group (except the last) must be exited at least once.

- Represent the string as a path of entrances/exits from groups
- The first unit of cost for exiting a group is not counted
- No permutation is repeated
- The total cost is bound by C: $C = L - n! - ((n-1)! - 1) - (n-1)$



Repeatedly adding symbols that incur no cost generates a **"zero cost cycle"** or **"group"**. There are $(n-1)!$ groups, each containing n permutations.

Example group, $n = 3$:



Results

n	L	Result	Naive Runtime	Basic Runtime	Higher Order Runtime
3	9	Sat	0.00 s	0.00 s	0.00 s
4	32	Unsat	27.65 s	0.02 s	0.09 s
4	33	Sat	5.66 s	0.05 s	0.09 s
5	152	Unsat	–	1410 s	25.70 s
5	153	Sat	–	5634 s	79.34 s

Discussion

From the results, we find that **incorporating more combinatorial properties** into an encoding of the problem can **improve the runtime** significantly.

We are much closer to attempting to solve encodings for superpermutations on six symbols.

As we progress to solving larger instances of the minimal superpermutation problem, the process will hopefully elucidate interesting insights to solving problems with high degree of symmetry using SAT solvers.

Future Work

Unique-permutation Conjecture: Assuming that a minimal superpermutation does not contain the same permutation twice provides significant speedup, but it is unproven.

Improve the Higher Order Cost Encoding: This encoding was implemented for the first time very recently. There are many opportunities for speed up in the encoding design.

Efficiently split the search space: Enable us to parallelize solving an encoding such that we do not repeat a significant amount of work. Essential for running larger formulae.