**Finding Minimum Inconsistent Subsets**

Solaman Huq

**Objective:**

The main goal of the study was to implement and analyze various algorithms used to find Minimal Inconsistent Subsets. The definition of a Minimal Inconsistent Subset involves some jargon that I will try to explain briefly: suppose we are given a set description, which in most practical applications is a Boolean circuit. Suppose that an instance of this circuit is not functioning properly, and we wish to find an explanation why. This explanation is a subset of constraints, or in this case logic gates within the circuit that we call “inconsistent”. A subset of logic gates is “inconsistent” for a given input if when we claim that its logic gates are “normal” and leave them alone, and claim that all other logic gatess are “abnormal” and force their output to an ideal, the final output is inconsistent with a normal output. More simply, the inconsistent subset is a subset of logic gates that is sufficient to explain the faulty output. Therefore, a Minimal Inconsistent Subset of these logic gates is a subset that is inconsistent, but removing any one item results in a consistent subset.

Once we noted that such subsets could be modeled with monotone Boolean functions, we started implemented efficient ways to generate such functions, which then allowed us to perform benchmarking on various algorithms to compute MIS, namely, Top Down, Bottom Up, Random, and Hitting Set Tree. Our benchmarking essentially asked “how many times must the algorithm call ‘isConsistent’, before it finds all Minimum Inconsistent Subsets?”

**Generating Monotone Boolean Functions:**

Before we could perform benchmarking on our algorithms, we first needed to find a way to consistently generate set descriptions and constraints. Since set descriptions could be modeled by Boolean circuits, we opted to generate them as Boolean functions ourselves. However, we also noted that the functions must be monotone; if a particular subset was “inconsistent”, any superset must also be “inconsistent” (naturally, blaming all of the circuits must lead to an output inconsistent with the normal output!) To this end, we concerned ourselves with monotone Boolean function generation.

While this is a hard problem, we found a simple way to generate all functions for a given input size. The implementation involves generation of MBF’s by level, specifically, that we start with the MBF with only one “inconsistent” subset, and that being of all logical gates being in the subset. Levels down would exclude one logical gate further. The algorithm is far from optimal, but still allowed us to generate MBF’s up to input size 5 within reasonable time (less than a second). For further explanation of the algorithm, see A First Attempt at Dedekind Numbers.

**Benchmarking Algorithms:**

Once we were able to generate MBF’s, our next step was to run benchmarks on the various algorithms we implemented. This essentially reduced to counting the number of times a given algorithm must ask the set description/MBF if a subset of constraints is “inconsistent”. Seeing as we had all 7581 MBF’s for input size n == 5, we calculated various statistics on each. Below are our results.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Input Size** | **Min** | **Max** | **Mean** | **Median** | **Mode** | **StdDev** |
| HST | 5 | 1 | 83 | 42 | 44 | 46 | 27.7139 |
| Bottom-Up | 5 | 1 | 32 | 21 | 20 | 26 | 8.2049 |
| Top-Down | 5 | 1 | 32 | 21 | 20 | 26 | 8.2019 |
| Random | 5 | 6 | 24 | 17 | 17 | 18 | 5.5522 |
| HST | 4 | 1 | 30 | 16 | 17 | 11 | 11.6972 |
| Bottom-Up | 4 | 1 | 16 | 11 | 11 | 11 | 5.4213 |
| Top-Down | 4 | 1 | 16 | 10 | 11 | 11 | 4.7958 |
| Random | 4 | 1 | 13 | 8 | 9 | 10 | 3.196465 |

We note that Hitting Set Tree proved to perform worse in comparison to the other algorithms, but what is interesting to note is that it performed more calls to “isConsistent” than there are subsets of constraints. However, we can reason this to a degree: as Hitting Set Tree utilizes Logarithmic Extraction as a black box all calls to “isConsistent” in Logarithmic Extraction cannot be used to make further deductions. We can see the possibility that, frequently, a set of calls to “isConsistent” will be redundant, while every call in the other algorithms is able to perform some elimination at each step. All this completely justifies the paper, A Hybrid Approach combining Black-Box and White-Box Reasoning!

The best performance being the Random algorithm also can be reasoned. Top-Down is not able to utilize elimination when “isConsistent” returns true, as in this instance, all above subsets cannot be MIS (as they are all supersets of an IS), but we are guaranteed to have visited all above subsets anyway! Bottom-Up has the reverse problem; as when “isConsistent” returns false, all bottom subsets cannot be IS (and therefore MIS), but we are guaranteed to have visited them anyway as well! The Random algorithm is able to bridge this gap by performing “isConsistent” on subsets that are, on average, in the middle of the power-set lattice of constraints.

**Moving Forward:**

Performing benchmarking on larger MBF’s would be challenging if we attempting to generate all of them per input size. As this ties directly to Dedekind Numbers, the number of MBF’s will grow exponentially, if not super exponentially! We would fare better to generate a set of, say, 100 random MBF’s per input size, and since the size of such MBF’s again grows exponentially, we can store these MBF’s implicitly to reduce on storage space.

Aside from that, there are always other algorithms to benchmark.