A Framework for Generating S-Box Circuits with Boyer–Peralta Algorithm-Based Heuristics, and Its Applications to AES, SNOW3G, and Saturnin

Artifact for TCHES 2025, Issue 1, Submission #84

1 Extended-BP-Framework

This paper provides guidance on using a tool for optimizing S-box circuits while preserving nonlinear gates and AND-depth. Once the `Extended_BP_Framework.zip` file is unzipped, all directories and files can be accessed.

2 How to Upload Circuits

To optimize a circuit, it must be uploaded as a Python file in the `code_target_imps` directory. The implementation of the circuit must be inserted at the position marked by the following comment in the file:

```
################### Here is your code !! ###################
```

The input and output variables must be listed as `x` and `y`, respectively. For further instructions, refer to the example file provided in the `code_target_imps` directory.

3 Options

- `-f` : Specifies the circuit filename to optimize (excluding the `.py` extension). [default : Imp_32ANDs]
- `-m` : Indicates the number of cores for multi-threading. Each core runs independently to optimize the target circuit. If set to 1, a single core is used. [default : 1]
- `-a` : Selects the BP-based algorithm to apply ('RNBP' or 'BPD'). [default : 'RNBP']
- `-d`, `-H` : Sets the depth limit. [default : 23]
- `-r` : Enable or disables random circuit modification mode (True or False). [default : False]

- `analyze` : Generates files by analyzing the result and log files.

4 Usage Examples

This section provides example commands along with explanations of their outcomes.

```
python main.py
```

This command performs the optimization process for the circuit specified in `code_target_imps/Imp_32ANDs.py`, utilizing eRNBP.

```
python main.py -f AES_depth16
```

The optimization process targets the circuit in `code_target_imps/AES_depth16.py`, utilizing eRNBP.

```
python main.py -f AES_depth16 -m 12 -a BPD -H 15
```

Here, the optimization focuses on the circuit in `code_target_imps/AES_depth16.py`, using eBPD with 12 cores and a depth limit of 15.

```
python main.py -f Ascon
```

In this case, the circuit in `code_target_imps/Ascon.py` is optimized with eRNBP.

```
python main.py -f Ascon -m 4 -a BPD -H 5 -r True
```

This command applies optimization to the circuit in `code_target_imps/Ascon.py` using cBPD with 4 cores, a depth limit of 5, and random circuit modification mode enabled.

```
python main.py -f Saturnin -m 8 -a RNBP
```

The optimization is applied to the circuit in `code_target_imps/Saturnin.py`, utilizing eRNBP on 8 cores.

```
python main.py analyze
```

The analysis processes all results stored in the `code_results` directory, generating a summary in `code_results/CODE_RESULTS.txt`. Log files in the `log` directory are reviewed, with changes in Dist saved to `log/view_Dist` and additional information stored in `log/view_all_Info`.

5 Generated Files by Our Tool

- The `code_results` directory includes the result `.py` files.
- The `code_formal` directory includes temporary `.py` files to extract XOR information.

- The `log` directory includes log files recording the optimization process.
- The `log/view_Dist` directory includes log files that record changes to *Dist*.
- The `log/view_all_Info` directory includes log files containing information other than *Dist*.

6 Notes on Analyzing the Results

Using the `analyze` option, in `code_results/CODE_RESULTS.txt`, the following metrics are analyzed: depth (with NOTs), depth (without NOTs), AND-depth, ANDs, ORs, XORs, and NOTs. When all NOT gates are combined into an XNOR gate, the latency of the circuit follows 'depth (without NOTs)' rather than 'depth (with NOTs)'.