# “Evolutionary Optimization of Compiler Flag Selection by Learning and Exploiting Flags Interactions“

Paper by: Unai Garciarena, Roberto Santana
Presentation by: Amanda Yao, Batuhan Akcay, Kevin Rodriguez-Siu

# Agenda

- Intro/Motivation
  - Why the paper is relevant to the class
  - Optimization flags
- GA, UMDA, EDA
  - Assumptions on the modeling of the problem
  - Basic workflow
  - How to use them for optimization flags
- Experiments
  - Benchmark dataset being used
  - Hardware setup
  - Comparison of 3 methods on different programs
  - Comparison of 3 types of operations in one program for EDA
- Conclusions and Future Works
  - Comments
  - Questions

# Introduction: Iterative Compilation

- Executable code can be significantly optimized at the time of compilation, to make significant gains in terms of time, energy, memory, etc. needed to solve a problem.
- Is it possible to use the information of this optimization to our advantage?
  - Modify the search space of our operations
  - Consider the statistical regularities present in the best solutions!
- **Iterative compilation:** Explore the space of possible optimization sequences until an acceptable solution is found.
- In this paper, the selection of a proper combination of compilation flags is analyzed in this way.

# Why Compilation Flags?

- Compilers provide a high flexibility for code quality by having a large set of flags to balance different criteria. But searching all over these compilation options manually is prohibitive due to the big quantity of terms ⇒ Genetic Algorithms
- We also need to consider the relationship and interactions between different flags! The presence of one flag might make another flag irrelevant, etc.
- Not only we need to search through a large space, but also this space is subject to relationships between variables.
- What if we try to capture them using probabilistic models?
- Proposal: ***Estimation of Distribution Algorithms (EDA)***

# A Basic Workflow for an Evolutionary Algorithm: Classical

**Algorithm 1: Evolutionary Algorithm**

1 Set $t \Leftarrow 0$. Generate an initial population $D_0$ of $N \gg 0$ random solutions.

2 **do**{

3 Select from population $D_t$ a set $D_{St}$ of $k \leq N$ points using truncation selection.

4 Generate a new population $D_{t+1}$ from $D_{St}$ applying the variator operator of choice.

5 $t \Leftarrow t+1$

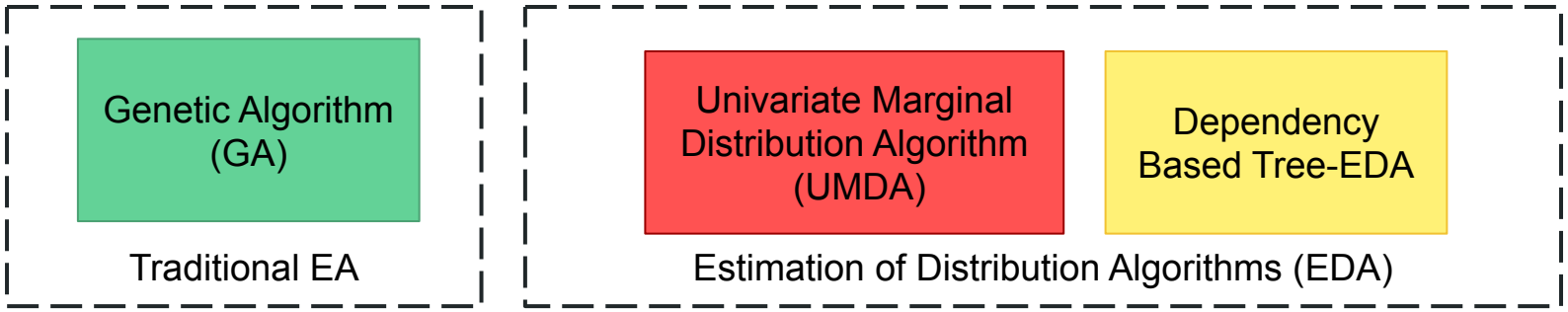
6 } **until** Termination Criteria are met.

We also need to consider:

- Nature of the variables in the population ⇒ Binary (a flag is either used or not).
- How we measure if a good solution has been found? ⇒ Fitness function and performance gain.

# Estimation of Distribution Algorithms - EDA

- Evolutionary Algorithms that learn a probabilistic model of the promising solutions in order to capture a description of relevant patterns shared by these solutions.
- Probabilistic Graphical Model (PGM) can store interactions between variables as dependency relationships and marginal probabilistic distributions.
- Will this model be able to capture information about the interactions between compiler flags?



# What do these EDA models represent?

Discrete random variables: $\mathsf{X} = (X_1, \ldots, X_n)$

Assigment to variables: $\mathsf{x} = (x_1, \ldots, x_n)$

Marginal probability for $\mathsf{X}_I = \mathsf{x}_I : p(\mathsf{x}_I)$

Conditional probability for $X_i = x_i$ given $X_j = x_j : p(x_i|x_j)$

Probability distribution conformal with a tree.

$$p_T(\mathsf{x}) = \prod_{i=1}^{l} p(x_i|pa(x_i))$$

Assume there are no parents?

$$p_u(\mathsf{x}) = \prod_{i=1}^{l} p(x_i)$$

*Assumptions* about the type of interaction between variables:

- UMDA assumes independency.
- Tree-EDA learns pair-wise dependency between variables.

If the problem has no interactions or they are weak enough, UMDA does good. If pairwise dependencies are essential for the problem, Tree-EDA can outperform other methods.

***How important are the relationships between compiler flags?***

# Benchmark Dataset

Mibench:

- 5 domains: automotive, consumer, network, office, security, telecomm
- Small and large datasets
- Used automotive domain

| Auto./Industrial |
| --- |
| basicmath |
| qsort |
| bitcount |
| susan (smoothing) |
| susan (edges) |
| susan (corners) |

# Experiment Setup

- GCC 58 flags
- Truncation Selection T=0.5
- Population N=100, max # generation n_gen=50
- Programmed in C++
- Hardware: i7 Pentium processor

**Experiment I:**

Determine the difference between the performance of GA, UMDA, and Tree-EDA.

| Auto./Industrial |
| --- |
| basicmath |
| qsort |
| bitcount |
| susan (smoothing) |
| susan (edges) |
| susan (corners) |

Table 1: Mibench instances used for the experimental framework.

```
double rad2deg(double rad)
{
      return (180.0 * rad / (PI));
}

double deg2rad(double deg)
{
      return (PI * deg / 180.0);
}
```

basicmath: cubic, isqrt, round, rad2deg

```
int CDECL bitcount(long i)
{
      i = ((i & 0xAAAAAAAAL) >>  1) + (i & 0x55555555L);
      i = ((i & 0xCCCCCCCCL) >>  2) + (i & 0x33333333L);
      i = ((i & 0xF0F0F0F0L) >>  4) + (i & 0x0F0F0F0FL);
      i = ((i & 0xFF00FF00L) >>  8) + (i & 0x00FF00FFL);
      i = ((i & 0xFFFF0000L) >> 16) + (i & 0x0000FFFFL);
      return (int)i;
}
```

bitcount

**Experiment I:**
Determine the difference between the performance of GA, UMDA, and Tree-EDA.

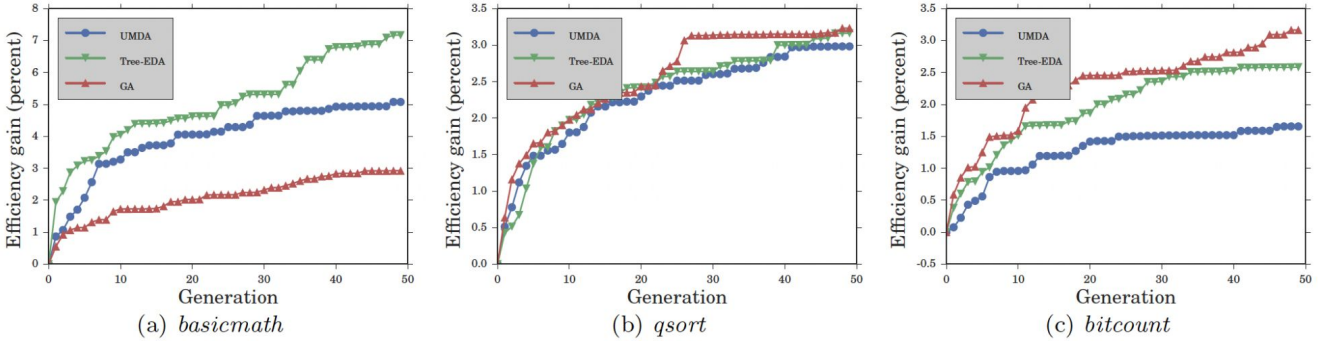
**Process**: run 30 executions of each algorithm and compared their performance in terms of the average best fitness reached by the algorithms in all the runs

**Evaluation Metric**: efficiency gain

$$eg = \frac{f_0 - f_t}{f_0}$$

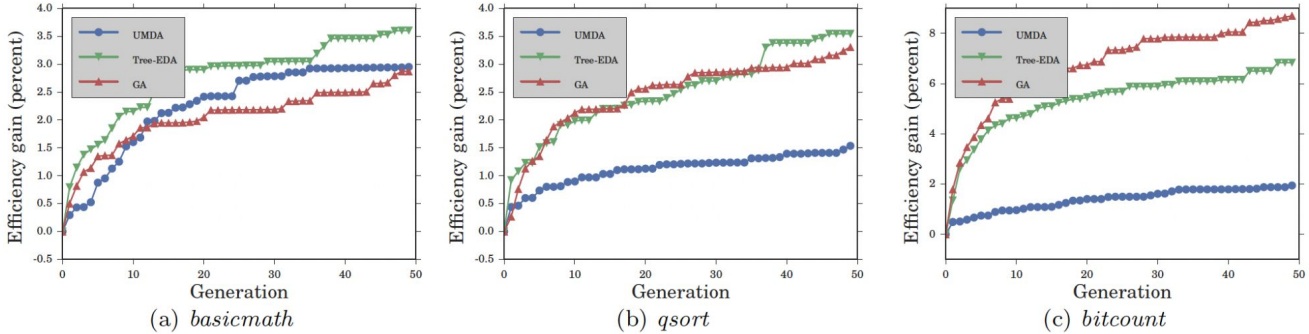
**Experiment I:**

Determine the difference between the performance of GA, UMDA, and Tree-EDA.




(a) *basicmath* (b) *qsort* (c) *bitcount*

Figure 1: Mean efficiency gain (percent) at each generation for the three EAs investigated (Small input).




(a) *basicmath* (b) *qsort* (c) *bitcount*

Figure 2: Mean efficiency gain (percent) at each generation for the three EAs investigated (Large inputs).

**Experiment I:**

Determine the difference between the performance of GA, UMDA, and Tree-EDA.

## Statistical Significance: (p=0.01)

| small | UMDA vs Tree | UMDA vs GA | Tree vs GA |
|---|---|---|---|
| basicmath | 1.555e-4 | 6.400e-12 | 1.818e-15 |
| qsort | - | - | - |
| bitcount | 4.610e-09 | 7.797e-11 | 1.620e-3 |
| **large** | **UMDA vs Tree** | **UMDA vs GA** | **Tree vs GA** |
| basicmath | 2.435e-5 | 2.300e-3 | 8.295e-8 |
| qsort | 3.300e-14 | 1.743e-13 | 4.316e-1 |
| bitcount | 3.375e-16 | 1.355e-16 | 8.945e-08 |

Table 2: Results of the statistical tests.

## Frequency:

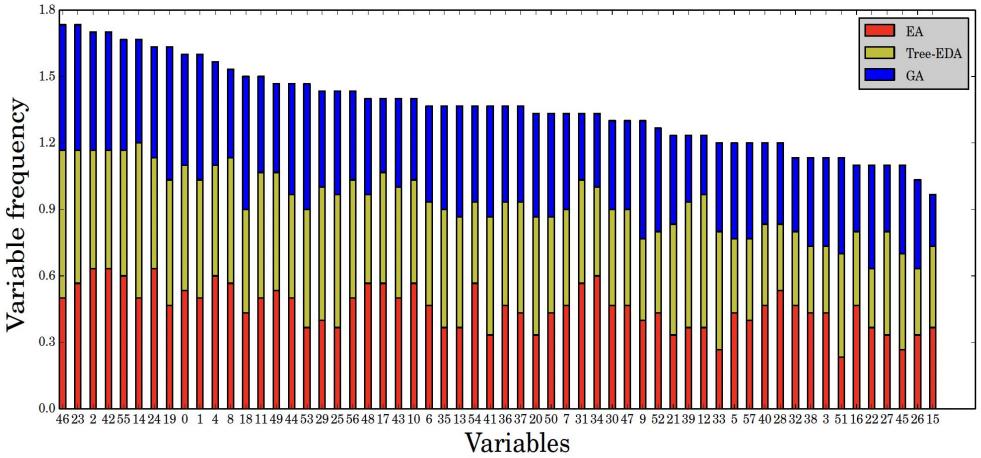



Figure 3: Frequency of the flags found by the three EAs in all the executions.

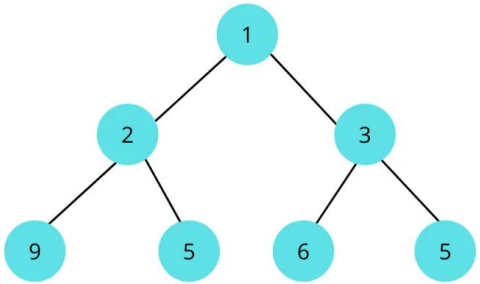
Most frequent flags were *fschedule-insns2*, *-ftree-salias*, and *-fkeep-static-consts*

least frequent flags were *-frerun-loop-opt*, *-ftree-ter*, and *-ftree-ch*.

**Experiment II:**
Are optimization flag interactions common to all programs or program-dependent?

**Process**: Run 30 executions of Tree-EDA on 3 different programs both with small and large input then compare the optimization flag interactions

**Evaluation Metric**: Frequency of optimization flag pairs in the Tree-EDA

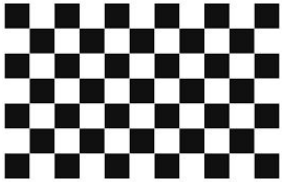



| Auto./Industrial |
|---|
| basicmath |
| qsort |
| bitcount |
| susan (smoothing) |
| susan (edges) |
| susan (corners) |

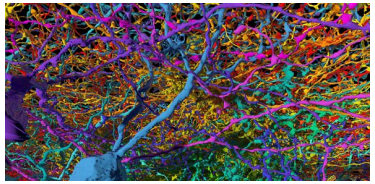
Table 1: Mibench instances used for the experimental framework.

**Experiment II:**

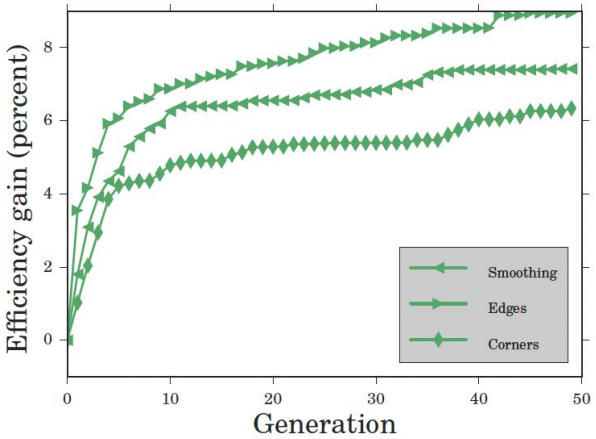
Are optimization flag interactions common to all programs or program-dependent?

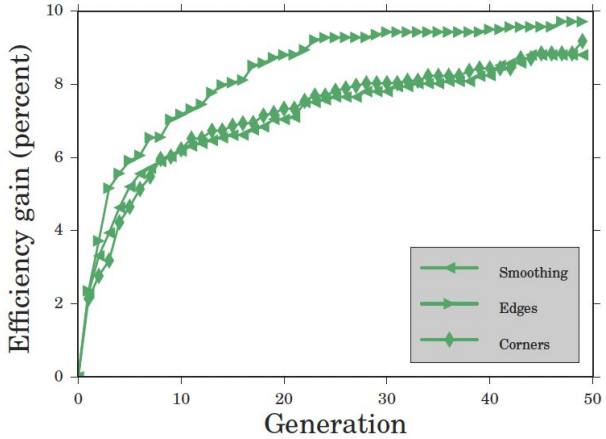


Small Input



Large Input



**Experiment II:**

Are optimization flag interactions common to all programs or program-dependent?

There exist pairs of flags that have a strong contribution to the optimization problem, but the number of such pairs are few.

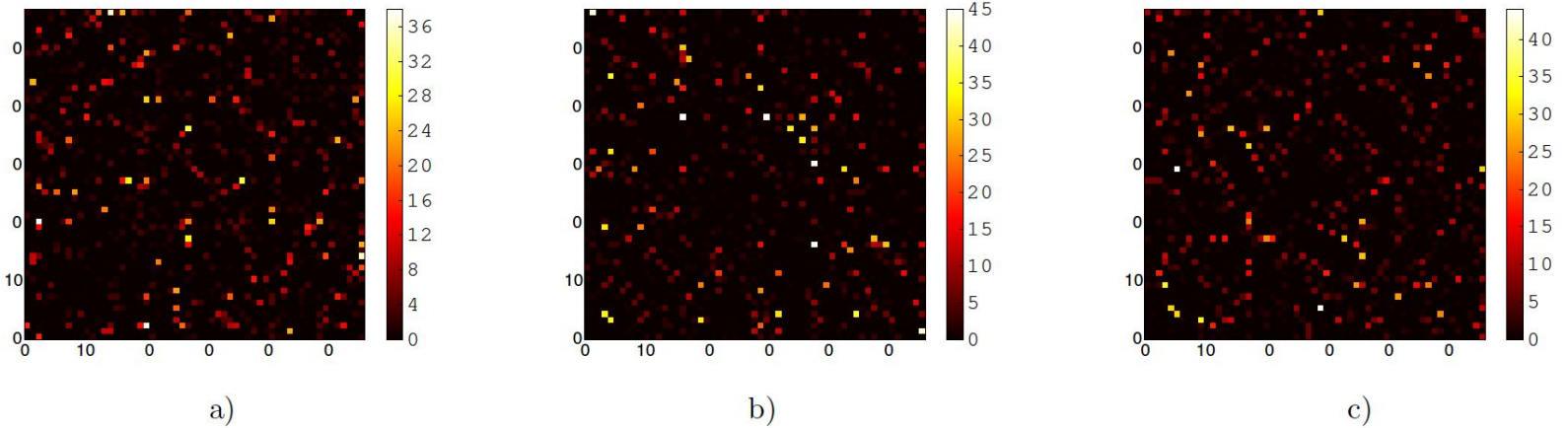


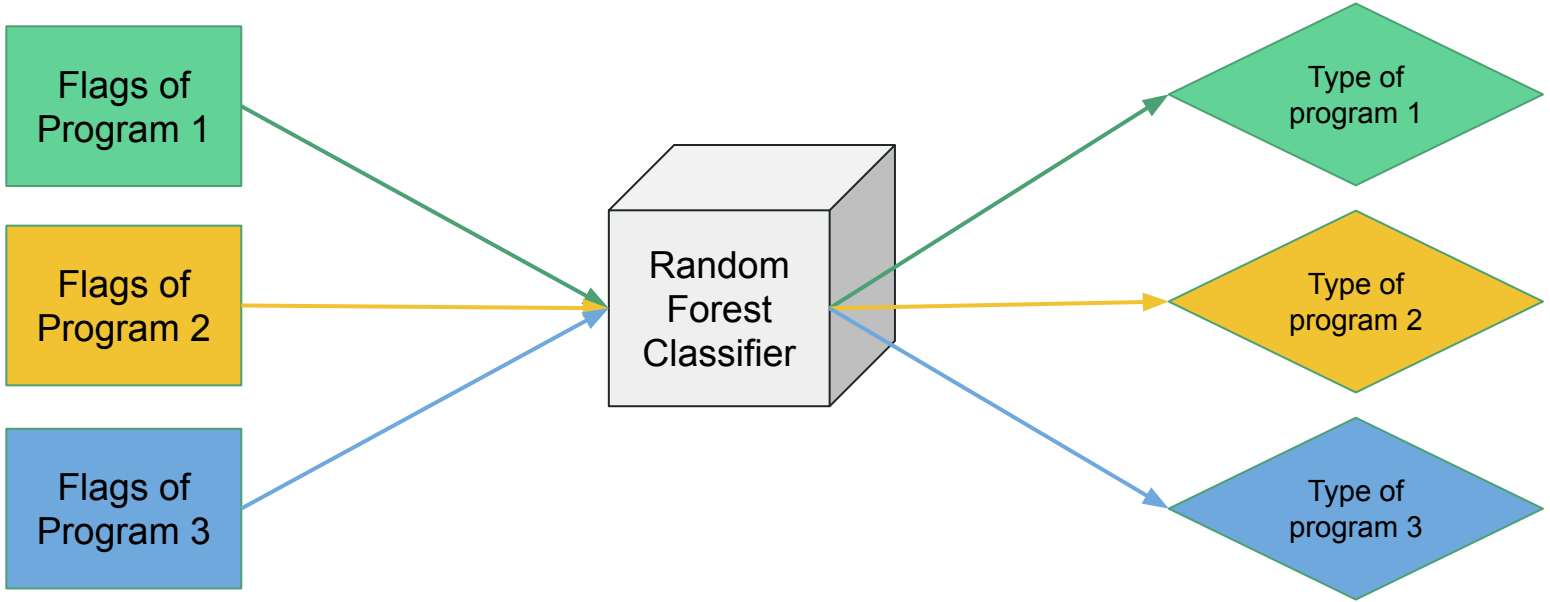

Figure 5: Most frequent interactions captured by Tree-EDA for the three different functionalities of program Susan when large images are considered. a) Smoothing, b) Edges, c) Corners

**Experiment II:**

Are optimization flag interactions common to all programs or program-dependent?

Given optimization flag outputs of Tree-EDA can we identify which type of program the Tree-EDA was run on?
(Types are smoothing, edges, and corners)

NO!



**Experiment II:**

Are optimization flag interactions common to all programs or program-dependent?

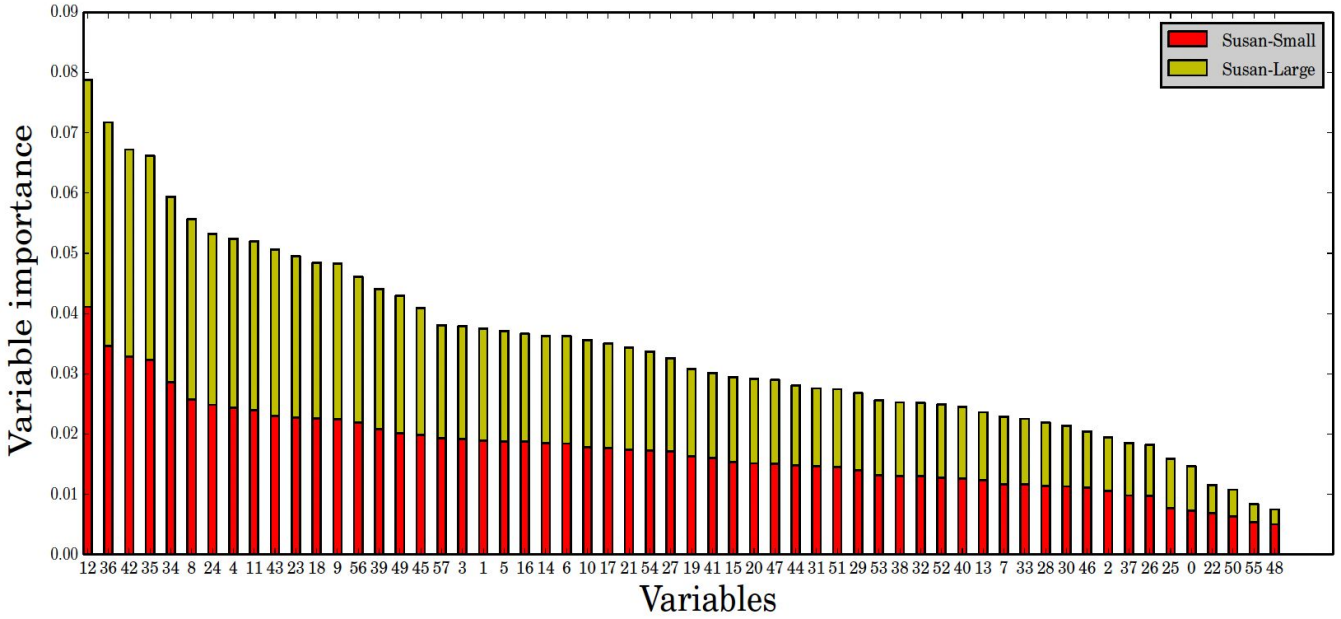



Figure 6: Feature importance found by the random forest classifier when classifying programs from the optimal flag sets.

## Conclusion:

- From other parts of presentation:
  - Searching best optimization flag combinations is a complex task due to:
    - Large search space
    - Relationship between each optimization flags
- From Experiment I:
  - Tree-EDA and GA perform better than UMDA for the choosing the flags for the optimization problem.
  - Most frequent flags were *fschedule-insns2*, *-ftree-salias*, and *-fkeep-static-consts*, least frequent flags were *-frerun-loop-opt*, *-ftree-ter*, and *-ftree-ch*.
- From Experiment II:
  - There exist pairs of flags that have a strong contribution to the optimization problem, but the number of such pairs are few.
  - Given optimization flag outputs of Tree-EDA we can not identify which type of program the Tree-EDA was run on
- Future works / ideas from the group:
  - Test current algorithms on additional datasets
  - Test other algorithms similar to the EDA on the same dataset to measure performance improvement


THANK
YOU