Graph-Based Algorithms for Boolean Function Manipulation
Introduction

- Boolean algebra forms a corner stone of Computer system & Digital design.

- So, we need efficient algorithms to represent & manipulate Boolean functions.

- Unfortunately many of the tasks involving with Boolean Functions requires solutions to NP-complete or co-NP-complete problems.
Most of the approaches require amount of computer time that grows exponentially with the size of the problem.

In practice there are some clever representation and manipulation techniques that avoids exponential computation.

Variety number of methods have been developed based on classical representation.
But these are also quite impractical.

There are some practical approaches which can be used at least for many functions without exponential size.
Disadvantages of some Practical approaches

- Still some functions require exponential size.
- Some functions have reasonable representation but its complement again leads to exponential.
- None of these representations are canonical forms.
Effects of these practical approaches

- Due to these characteristics most of these programs that process a sequence of operations on Boolean functions have erratic behavior.

- They may proceed at regular pace but suddenly blow up.
What is my project?
Introducing a new method

• Our method resembles the Binary decision diagram notation introduced by Lee and further popularized by Akers.

• However, we place further restrictions on the ordering of decision variables in the vertices.

• These restriction enable us to develop algorithms in a more efficient manner.
Advantages

• Most commonly encountered functions have reasonable representations.

• Performance of program degrades slowly if at all.

• Representation in terms of reduced graphs have ‘Canonical forms’.
Disadvantages

• At the start of the process we must choose some ordering of the system inputs as arguments to all of the functions to be represented.

• For some functions ordering of the inputs are highly sensitive.

• The problem of computing an ordering that minimizes the size of graph is itself a co NP-complete problem.

• For other class of problems our method seems practical only under restricted conditions.
Basic concepts of our work
Definition 1: A function graph is rooted, directed graph with vertex set $V$ containing two types of vertices. A nonterminal vertex $v$ has as attributes an argument index $\text{index}(v) \in \{1, \ldots, n\}$ and two children $\text{low}(v), \text{high}(v) \in V$. A terminal vertex $v$ has as attribute a value $\text{value}(v) \in \{0, 1\}$.
Definition 2: A function $G$ having root vertex $v$ denotes a function $f_v$ defined recursively as

1) If $v$ is a terminal vertex:
   - If $\text{value}(v) = 1$, then $f_v = 1$.
   - If $\text{value}(v) = 0$, then $f_v = 0$.

2) If $v$ is nonterminal vertex with $\text{index}(v) = i$, then $f_v$ is the function.
   
   $f_v(x_1, \ldots, x_n) = x_i \cdot f_{\text{low}(v)}(x_1, \ldots, x_n) + x_i \cdot f_{\text{high}(v)}(x_1, \ldots, x_n)$. 
Definition 3: Function graphs G and G’ are isomorphic if there exists a one-to-one function \( \sigma \) from the vertices of G onto the vertices of G’ such that any vertex \( v \) if \( \sigma(v) = v' \), then either both \( v \) and \( v' \) are terminal vertices with \( \text{value}(v) = \text{value}(v') \), or both \( v \) and \( v' \) are non terminal vertices with \( \text{index}(v) = \text{index}(v'), \sigma(\text{low}(v)) = \text{low}(v'), \) and \( \sigma(\text{high}(v)) = \text{high}(v') \).
**Definition 4:** For any vertex $v$ in a function graph $G$, the subgraph rooted by $v$ is defined as the graph consisting of $v$ and all its descendants.
Definition 5: A function graph \( G \) is reduced if it contains no vertex \( v \) with \( \text{low}(v) = \text{high}(v) \), nor does it contain distinct vertices \( v \) and \( v' \) are isomorphic.

Theorem: For any boolean function \( f \), there is a unique reduced function graph denoting \( f \) and any other function denoting \( f \) contains more vertices.
Example Functions

![Graphs of functions](Image)

**Fig. 1.** Example function graphs.
Ordering dependence

Fig. 2. Example of argument ordering dependence.
Inherently complex functions
## Summary of Basic Operations

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Result</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduce</td>
<td>$G$ reduced to canonical form</td>
<td>$O(</td>
</tr>
<tr>
<td>Apply</td>
<td>$f_1(op)f_2$</td>
<td>$O(</td>
</tr>
<tr>
<td>Restrict</td>
<td>$f</td>
<td>_{x_i=b}$</td>
</tr>
<tr>
<td>Compose</td>
<td>$f_1</td>
<td>_{x_i=f_2}$</td>
</tr>
<tr>
<td>Satisfy-one</td>
<td>some element of $S_f$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Satisfy-all</td>
<td>$S_f$</td>
<td>$O(n \cdot</td>
</tr>
<tr>
<td>Satisfy-count</td>
<td>$</td>
<td>S_f</td>
</tr>
</tbody>
</table>
Conclusion

• Given any graph representing a function, we can reduce it to reduced graph in nearly linear time.

• We could represent a set of functions by by a single graph with multiple roots.
Questions ?
Thank you