Introduction

- Binary Decision Diagram
  - A data structure used for encoding solutions of Boolean functions.
  - ZDD is a modified version of BDD, used for encoding set of Boolean combinations, i.e. implicit Boolean function.
  - A rooted, directed acyclic graph.
  - Two terminal nodes:
    - 0- empty set: 0.
    - 1- set of empty set: (0).
  - Non-leaf node u is represented by a three tuple (l, i, h):
    - l: The Boolean variable represented by u.
    - i: Low edge "<" variable i doesn't exist or has value 0.
    - h: High edge "=" variable i exists or has value 1.

How the encoding works

- Inputs:
  - A Boolean function:
    - \( f(x_1, x_2, y_2) = (x_1 \equiv y_1) \land (x_2 \equiv y_2) \)
  - An ordering:
    - \( x_1 < y_1 < x_2 < y_2 \)
  - Start from \( x_1 \), replace it with either 0 or 1:
    - \( f(0, 0, 1, 1, 2, 2) = (0 \equiv y_1) \land (x_2 \equiv y_2) \)
    - \( f(1, 1, 1, 1, 2, 2) = (1 \equiv y_1) \land (x_2 \equiv y_2) \)
  - Continue with other variables, until all possible combinations are evaluated:
    - \( f(0, 0, 0, 0) = (0 \equiv 0) \land (0 \equiv 0) = 1 \)
    - \( f(1, 1, 0, 0) = (1 \equiv 0) \land (0 \equiv 0) = 0 \)

- A solution is a path from the root to terminal 1.

Application

1. Solving graph-coloring problem

   - Fix a vertex ordering
   - Build ZDD to represent maximum independent sets

   **Problem:** Color the graph so that no adjacent vertices share the same color.
   **Solution:** A ZDD where a path from root to terminal represents a set of vertices that can have the same color.

2. Solving shortest-path problem

   - Fix an edge ordering
   - Build ZDD to characterize all paths from S to T
   - Return the path with the minimum weight

   **Problem:** Given a directed graph with source and sink. T.
   **Solution:** A ZDD that encodes all possible paths from S to T.

Algorithm: MakeIndSetZDD

- **input:** A set \( U = \{u_1, u_2, \ldots, u_k\} \) of uncolored vertices such that \( u_i < u_{i+1} \) with respect to the vertex ordering on \( V \), and a “current index” \( i \)
- **output:** The root node of a ZDD characterizing all the maximal independent sets in \( G[U] \) that can be formed with vertices in \( \{u_1, u_2, \ldots, u_k\} \)

1. If \( G[U] \) cannot be covered by taking all vertices in \( \{u_1, u_2, \ldots, u_k\} \): return 0
2. If \( U = \emptyset \): return 1
3. \( U'_i = U - u_i - N(u_i) \) \( \{ \text{Use vertex } u_i; \text{ remove it and its neighbors from } U \} \)
4. \( h = \min \{f \mid u_i > u_i, \text{ and } u_i \in U'_i \} \) or \( |U'_i| + 1 \) if no such \( y \) exists
5. \( b_h = \text{MakeIndSetZDD} (U'_i, h) \)
6. \( b_i = \text{MakeIndSetZDD} (U, i + 1) \)
7. If \( b_h = 0 \): return \( b_i \)
8. \{ \text{Use reverse lookup table} \}
9. If \( e \in E \) \( x \), \( v(a) = i, b(a) = b_i \) and \( h(a) = b_h \): return a
10. else: return \( ZDZ . i n s e r t ( h, b_i ) \)

Findings

- The size of ZDD and the time it takes to construct the ZDD all depend on vertex or edge ordering of the original graph.
- Four different edge ordering rules:
  - Order edge by vertex indegree.
  - Order edge by vertex outdegree.
  - Order edge by path from source to sink.

Example: Given a directed graph with 10 vertices and 45 edges, build ZDD using MakePathZDD algorithm to find all paths:

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Adjacent vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7, 4, 8</td>
</tr>
<tr>
<td>2</td>
<td>48, 765, 613</td>
</tr>
<tr>
<td>3</td>
<td>418, 765</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>47, 168</td>
</tr>
<tr>
<td>6</td>
<td>48, 760, 1</td>
</tr>
<tr>
<td>7</td>
<td>048</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>9</td>
<td>816, 43132</td>
</tr>
</tbody>
</table>

Use DFS to find several paths first and assign priority order to the edges that are on these paths:

<table>
<thead>
<tr>
<th>Source</th>
<th>Sink</th>
<th>Performance (ZDD size/time)</th>
<th>Original Performance (ZDD size/time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0</td>
<td>1800 (0.05s)</td>
<td>2300 (0.02s)</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1020 (0.02s)</td>
<td>1520 (0.01s)</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1160 (0.01s)</td>
<td>1370 (0.02s)</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>940 (0.01s)</td>
<td>182 (0.02s)</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1420 (0.01s)</td>
<td>1850 (0.01s)</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1520 (0.01s)</td>
<td>1810 (0.02s)</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1380 (0.01s)</td>
<td>1410 (0.02s)</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1280 (0.01s)</td>
<td>940 (0.01s)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1340 (0.01s)</td>
<td>1760 (0.02s)</td>
</tr>
</tbody>
</table>

Future Work

- Further research into the relation between edge ordering and the size of ZDD.
- Find out how the structure of the graph affects ZDD size.

Special thanks to:

- Reference