Background

Sying-Jyan Wang
Dept. Computer Science
National Chung-Hsing University

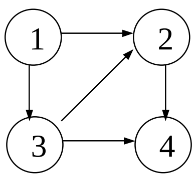
Outline

- Graph algorithms
- Functional expression
- Binary Decision Diagram (BDD)

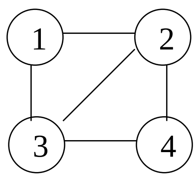
Elementary Graph Theory

- A graph $G = (V,E)$
  - V: a finite non-empty set of vertices
  - E: a set of edges (v,w), where $v,w \in V$
- Directed vs. Undirected Graph
  - If the pair (v,w) is ordered, the graph is **directed**.
    - v is the tail and w is the head.
  - Otherwise, the graph is **undirected**.
- Tree: a graph with no undirected cycles

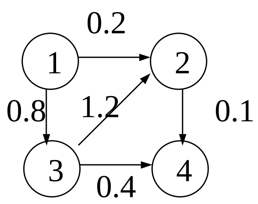
Graph Examples



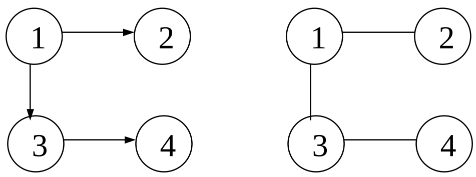
Directed, Acyclic (DAG)



Undirected



Weighted (Labeled)

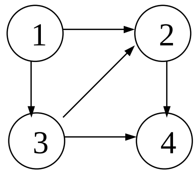


Trees have no undirected cycles

Trees

Graph Representation

- Adjacent list
  - Topological representation of the graph with vertices, edges, and associated properties
- Adjacent matrix



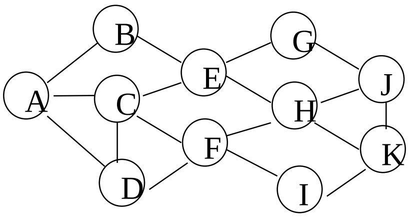
$$
\begin{array}{c c}
 & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}
\end{array}
$$

Graph Search Algorithms

- Search a given graph for special properties
- Two common techniques
  - Depth first search (DFS)
    - Implemented with a stack (implicitly)
  - Breadth first search
    - Implemented with a queue

Depth First Search

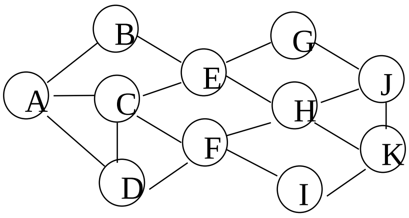
```
dfs(G, u) {
    foreach (v∈V)
            v.visited = false;
    dfs_search(u);
}
dfs_search(v) {
    v.visited = true;
    foreach_edge (v,w)
            if (w.visited == false)
                        dfs_search(w);
}
```



DFS order:
A,D,F,I,K,J,G,E,B,C,H

Breadth First Search

```
bfs(G, u) {
    foreach (v∈V)
      v.visited = false;
    u.visited = true;
    bfs_search(u);
}
bfs_search(v) {
    foreach_edge (v,w) {
      if (w.visited == false) {
          enqueue(Q,w); //Q: a queue
          w.visited = true;
      }
    while ((w=dequeue(Q)) != NIL)
          bfs_search(w);
}
```



BFS order:
A,D,C,B,F,E,I,H,G,K,J

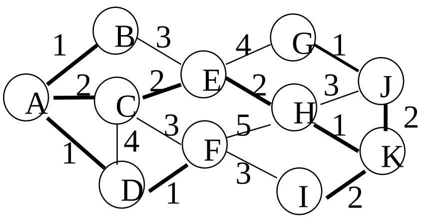
Cycle Detection

- Use depth first search
- Can be modified to output the vertices in the cycle

```
dfs(G, u) {
    foreach (v∈V)
        v.visited = false;
    dfs_search(u);
}
dfs_search(v) {
    v.visited = true;
    foreach_edge (v,w)
        if (w.visited == false)
            dfs_search(w);
        else
            printf("Cycle Detected")
}
```

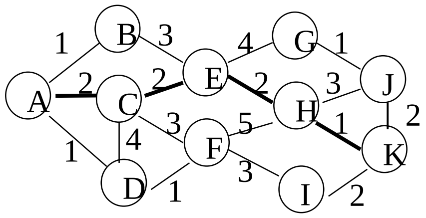

Minimum Spanning Trees

- Given a graph G=(V,E), a subgraph G'=(V',E') is a minimum spanning tree if
  - $V' = V$
  - $E' \subseteq E$
  - G' is a tree
  - The weight associated with E' is minimum



Shortest Path

- Find a path between two vertices with the minimum path weight



Related Problems

- Vertex Cover of an undirected graph $G=(V,E)$
  - A subset of V such that each edge in E has at least one end point in that subset.
- Graph Coloring of an undirected graph $G=(V,E)$
  - A labeling of V such that no each edge in E has two end point with the same label.
- Clique
  - A clique is a complete subgraph
  - Clique partition
  - Clique covering

Combinatorial Optimization

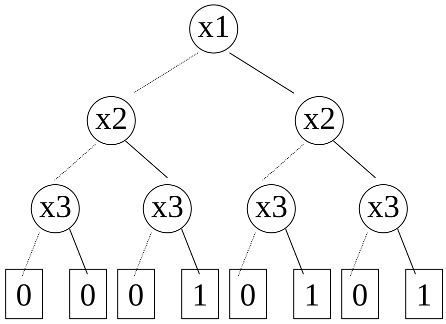
- Search the solution space for the optimal solution
- In many cases, they are intractable
  - NP hard or NP-complete
- Typical algorithms
  - Branch-and-bound algorithms
  - Linear and integer programming
  - Dynamic programming
  - Greedy algorithms

Binary Decision Diagram (BDD)

- A compact way to represent Boolean functions
  - Decision graph structure
  - Affected by variable order
- Algorithms
  - Basic operations
    - Restriction
    - If-then-else
  - Derived operations

Decision Tree Structures

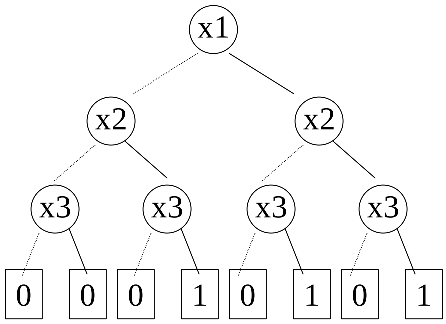
- Vertices represent decision
- Follow a broken line for value 0
- Follow a broken line for value 1
- Function value is determined by the leaf value



Ordered BDD (OBDD)

- A decision tree in which
  - Terminal node v
    - Attribute
      - value(v) = 0
      - value(v) = 1
  - Nonterminal node:
    - index(v) = i
    - Two children
      - low(v)
      - high(v)
    - index(v) < index(*low*(v)), index(v) < index(*high*(v))

$f = x_2x_3 + x_1x_3$

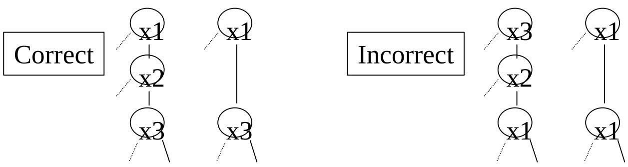


BDD Structure (Cnt'd)

- A BDD has a vertex v as the root corresponds to the function F_v:
  - If v is a terminal node:
    - If value(v) = 1, then $F_v = 1$
    - If value(v) = 0, then $F_v = 0$
  - If v is a nonterminal node (with index(v) = i)
    - $F_v(x_1, x_2, \ldots, x_n) = x_i' F_{low(v)}(x_1, x_2, \ldots, 0, \ldots, x_n) + x_i F_{high(v)}(x_1, x_2, \ldots, 1, \ldots, x_n)$
    - This is the Shannon's expansion w.r.t. x_i

Variable Ordering

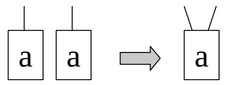
- Assign an arbitrary order to variables
- Variables appear in ascending order along all paths
- Example: assume $x_1 < x_2 < x_3$

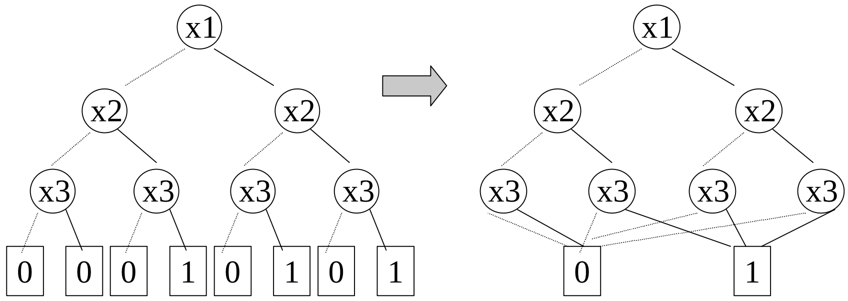


- Properties:
  - No conflicting variable assignment along path
  - Simplifies manipulation

Reduction

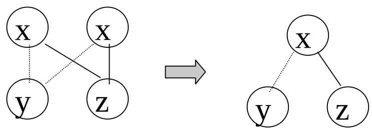
- Reduction rule 1:
  merge equivalent leaves

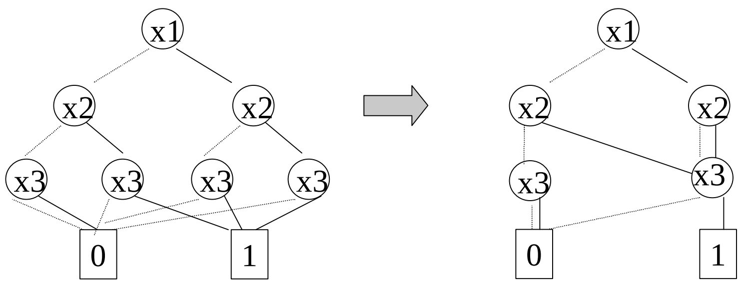




Reduction (Cnt'd)

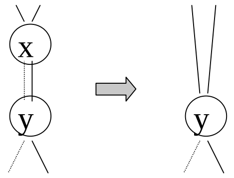
- Reduction Rule 2:
  merge isomorphic nodes

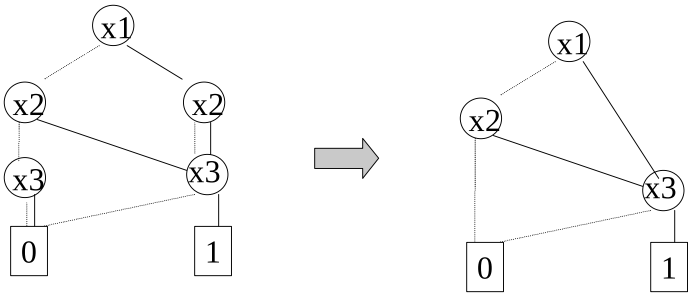




Reduction (Cnt'd)

- Reduction Rule 3:
  eliminate redundant nodes



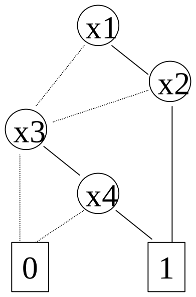


ROBDD

- An ROBDD is an OBDD in which
  - No vertex *v* with *low*(*v*) = *high*(*v*)
  - No pair {*u, v*} such that the subgraphs rooted in *v* and in *u* are isomorphic
- Canonical representation of Boolean function for the given ordering
  - Two functions are equivalent iff graphs are isomorphic
    - Can be tested in linear time
  - Desirable properties: simplest form is canonocal

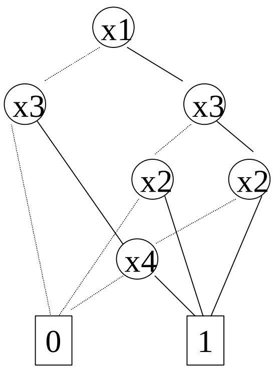
Size of an OBDD

- The size of an OBDD depends on the ordering of variables
  - Ex: $x_1x_2+ x_3x_4$, with $x_1 < x_2 < x_3 < x_4$



Size of an OBDD (Cnt'd)

- Another order $x_1 < x_3 < x_2 < x_4$



Sample Function Classes

- Rule of thumb
  - Many tasks have reasonable OBDD representations
  - Algorithms remain practical for up to 1000-node OBDD
  - Heuristic ordering methods generally satisfactory

Function Class	Best	Worst	Ordering Sensitivity
ALU	Linear	Exponential	High
Symmetric	Linear	Quadratic	None
Multiplication	Exponential	Exponential	Low

Symbolic Manipulation

- Strategy
  - Represent data as set of OBDDs
    - With identical variable orderings
  - Express solution method as sequence of symbolic operations
  - Implement each operation by OBDD manipulation
- Algorithm Properties
  - Arguments are OBDDs with identical variable orderings
  - Result in OBDD with same ordering
  - “Closure Properties”
- Two Basic operations: 1. Restriction, 2. If-Then-Else

Restriction Operation

- Concept
  - Set an argument to a constant k (0 or 1)
  - Also called Cofactor operation
- Implementation
  - Depth-first traversal
  - Complexity: near-linear in argument graph size

Restriction Algorithm

- Algorithm

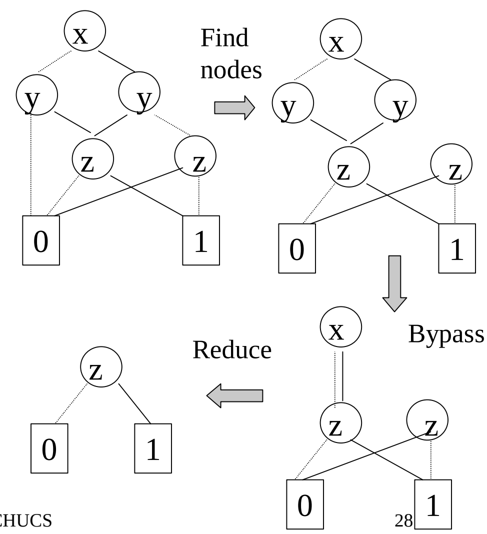
  Restrict (F, x, k)

  Bypass nodes with variable x;

  // choose high child for k=1

  // choose low child for k=0

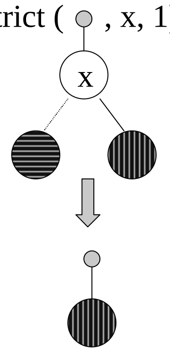
  Reduce result
- Ex. Restrict variable y to 1



Special Cases for Restriction

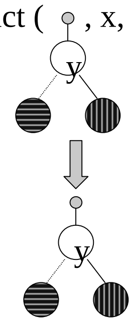
- Case 1: Restrict on root node variable

  Restrict (, x, 1)



- Case 2: Restrict on variable other than root node (e.g. x < y)

  Restrict (, x, 1)



If-Then-Else Operation

- Basic technique for building OBDD from network of formula
- Argument I, T, E
  - Functions over variables x
  - Represented as OBDDs
- Result
  - OBDD representing composite function
  - IT+I'E
- Implementation
  - Combination of depth-first traversal and dynamic programming
  - Worst case complexity: product of argument graph size`

If-Then-Else Algorithm

- Recursive formulation

  $ITE(I, T, E) = x\ ITE(I|_{x=1}, T|_{x=1}, E|_{x=1}) +$
  $x'\ ITE(I|_{x=0}, T|_{x=0}, E|_{x=0})$
- General Algorithm
  - Select top root variable x of I, T, and E
  - Compute restrictions
  - Apply recursively to get results *low* and *high*
- Terminal Conditions

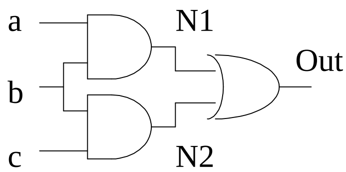
  1. I = 1 ➔ Return T; 2. I = 0 ➔ Return E;
  3. T = 1, E = 0 ➔ Return I; 4. T = E ➔ Return T;

Derived Algebraic Operations

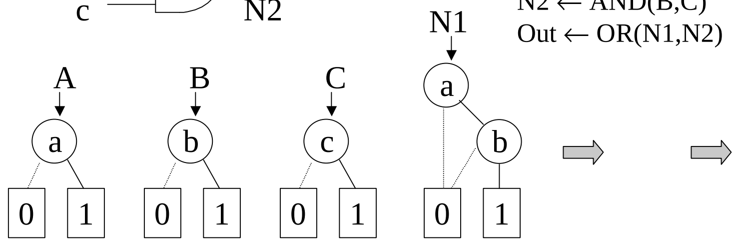
- Other common operations can be expressed in terms of if-then-else
  - AND(f, g) = ITE(f, g, 0)
  - OR(f, g) = ITE(f, 1, g)
  - EXOR(f, g) = ITE(f, g', g)

Generating OBDD from Network

- Goal: represent outputs of gate network as BDDs



A ← new_var("a")
B ← new_var("b")
C ← new_var("c")
N1 ← AND(A,B)
N2 ← AND(B,C)
Out ← OR(N1,N2)



Derived Operations

- Efficiency
  - Maintain computed table to increase efficiency
  - Worst case complexity product pf graph sizes for I, T, E
- Derived operations
  - Express as combination of if-then-else and restrict
  - Preserve closure property
    - Result in an OBDD with the same variable ordering