Overview of Decision Diagrams for Optimization

J. N. Hooker
Carnegie Mellon University

IOS Conference 2016
Today’s Session

• **Overview** of decision diagrams for optimization  
  – JH

• Decisions diagrams for **sequencing and scheduling**  
  – Andre Cire

• Decision diagram **decompositions**  
  – David Bergman
Decision Diagrams

• **Used in computer science and AI for decades**
  – Logic circuit design
  – Product configuration

• **A new perspective** on optimization
  – Constraint programming
  – **Discrete optimization**
Decision Diagrams

• Advantages:
  – No need for **inequality** formulations.
  – No need for **linear** or **convex** relaxations.
  – New approach to solving **dynamic programming** models.
  – Very effective **parallel** computation.
  – Ideal for **postoptimality** analysis

• Disadvantage:
  – Developed only for **discrete, deterministic** optimization.
  – …so far.
Outline

• Decision diagram basics
• Optimization with exact decision diagrams
• A general-purpose solver that scales up
  – Relaxed decision diagrams
  – Restricted decision diagrams
  – Dynamic programming model
  – A new branching algorithm
  – Computational performance
• Modeling the objective function
  – Inventory management example
• References
Decision Diagram Basics

- Binary decision diagrams encode Boolean functions

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Lee (1959), Akers (1978)
Decision Diagram Basics

• Binary decision diagrams encode Boolean functions
  – Historically used for circuit design & verification

Bryant (1986), etc.
Decision Diagram Basics

- Binary decision diagrams encode Boolean functions
  - Historically used for circuit design & verification
  - Easily generalized to multivalued decision diagrams
Reduced Decision Diagrams

- There is a unique reduced DD for any given constraint.
  - Once the variable ordering is specified.

- The reduced DD can be viewed as a branching tree with redundancy removed.
  - Superimpose isomorphic subtrees.
  - Remove redundant nodes.

Bryant (1986)
Branching tree for 0-1 inequality

\[ 2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7 \]

1 indicates feasible solution, 0 infeasible
Branching tree for 0-1 inequality

\[ 2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7 \]

Remove redundant nodes…
Superimpose identical subtrees...
Superimpose identical subtrees…
Superimpose identical leaf nodes…
as generated by software
Reduced Decision Diagrams

• Reduced DD for a knapsack constraint can be surprisingly small…

The 0-1 inequality

\[
300x_0 + 300x_1 + 285x_2 + 285x_3 + 265x_4 + 265x_5 + 230x_6 + 230x_7 + 190x_8 + 200x_9 + \\
400x_{10} + 200x_{11} + 400x_{12} + 200x_{13} + 400x_{14} + 200x_{15} + 400x_{16} + 200x_{17} + 400x_{18} \leq 2700
\]

has 117,520 minimal feasible solutions (or minimal covers)

But its reduced BDD has only 152 nodes…
Optimization with Exact Decision Diagrams

- Decision diagrams can represent feasible set
  - Remove paths to 0.
  - Paths to 1 are feasible solutions.
  - Associate costs with arcs.
  - Find longest/shortest path

Hadžić and JH (2006, 2007)
Stable Set Problem

Let each vertex have weight $w_i$

Select nonadjacent vertices to maximize $\sum_i w_i x_i$
Exact DD for stable set problem
Exact DD for stable set problem
Paths from top to bottom correspond to the 9 feasible solutions.
For objective function, associate weights with arcs.
For objective function, associate weights with arcs

Optimal solution is *longest path*
For objective function, associate weights with arcs

Optimal solution is longest path
Exact DD Compilation

• Build an exact DD by associating a state with each node.
  • Merge nodes with identical states.
Exact DD for stable set problem

To build DD, associate state with each node
Exact DD for stable set problem

To build DD, associate state with each node
Exact DD for stable set problem

To build DD, associate state with each node

\[
\begin{align*}
X_1 & \rightarrow \{12345\} \rightarrow X_2 \\
X_3 & \rightarrow \{2345\} \\
X_4 & \rightarrow \{34\}
\end{align*}
\]
Exact DD for stable set problem

To build DD, associate **state** with each node
Exact DD for stable set problem

Merge nodes that correspond to the same state.
Exact DD for stable set problem

Merge nodes that correspond to the same state
Exact DD for stable set problem

To build DD, associate state with each node
Exact DD for stable set problem

Resulting DD is not necessarily reduced (it is in this case).
A General-Purpose Solver

• The decision diagram tends to grow exponentially.
• To build a **practical solver:**
  – Use limited-width **relaxed** decision diagrams to bound the objective value.
  – Use limited-width **restricted** decision diagrams for primal heuristic
  – Use a **recursive dynamic programming model.**
  – Use **novel branching scheme** within relaxed decision diagrams.
Relaxed Decision Diagrams

- A relaxed DD represents a superset of feasible set.
  - Shortest (longest) path length is a bound on optimal value.
  - Size of DD is controlled.
  - Analogous to LP relaxation in IP, but discrete.
  - Does not require linearity, convexity, or inequality constraints.

Andersen, Hadžić, JH, Tiedemann (2007)
To build relaxed DD, merge some additional nodes as we go along.
To build relaxed DD, merge some additional nodes as we go along.
To build \textbf{relaxed} DD, merge some additional nodes as we go along.

Take the \textbf{union} of merged states.
To build relaxed DD, merge some additional nodes as we go along.

Take the union of merged states.
To build **relaxed** DD, merge some additional nodes as we go along.

Take the **union** of merged states.
To build **relaxed** DD, merge some additional nodes as we go along.

Take the **union** of merged states.
Width = 2

Represents 11 solutions, including 9 feasible solutions.
Represents 11 solutions, including 9 feasible solutions.

Width = 2

Longest path (90) gives bound on optimal value (70).
Relaxed Decision Diagrams

• Original application: enhanced propagation in constraint programming
  – In multiple alldiff problem (graph coloring), reduced 1 million node search trees to 1 node.

Andersen, Hadžić, JH, Tiedemann (2007)
Relaxed Decision Diagrams

- Wider diagrams yield tighter bounds
  - But take longer to build.
  - Adjust width dynamically.

Bergman, Ciré, van Hoeve, JH (2013)
Relaxed Decision Diagrams

- DDs vs. CPLEX bound at root node for max stable set problem
  - Using CPLEX default cut generation
  - DD max width of 1000.
  - DDs require about 5% the time of CPLEX

Bergman, Ciré, van Hoeve, JH (2013)
Restricted Decision Diagrams

- A **restricted** DD represents a **subset** of the feasible set.
- Restricted DDs provide a basis for a **primal heuristic**.
  - Shortest (longest) paths in the restricted DD provide good feasible solutions.
  - Generate a **limited-width** restricted DD by deleting nodes that appear unpromising.

Bergman, Ciré, van Hoeve, Yunes (2014)
Set covering problem

\[ x_1 + x_2 + x_3 \geq 1 \]
\[ x_1 + x_4 + x_5 \geq 1 \]
\[ x_2 + x_4 + x_6 \geq 1 \]

52 feasible solutions.

Minimum cover of 2, e.g. \( x_1, x_2 \)
Restricted DD of width 4

Several shortest paths have length 2.

All are minimum covers.

41 paths (< 52 feasible solutions)
Several shortest paths have length 2.

All are minimum covers.

In this case, restricted DD delivers optimal solutions.

41 paths (< 52 feasible solutions)
Optimality gap for set covering, $n$ variables

Restricted DDs vs Primal heuristic at root node of CPLEX

![Graph showing the comparison between Restricted DDs and Primal heuristic for different values of $n$. The graph plots the Average Optimality Gap (%) against $n$. The y-axis ranges from 35 to 60, and the x-axis from 0 to 4000. The graph indicates that the Optimality Gap for Restricted DDs is consistently lower than that of the Primal heuristic for all values of $n$.](image-url)
Computation time

Restricted DDs vs Primal heuristic at root node of CPLEX (cuts turned off)
Dynamic Programming Model

- Formulate problem with **dynamic programming** model.
  - Rather than constraint set.
  - Problem must have **recursive** structure
  - But there is great **flexibility** to represent constraints and objective function.
  - Any function of **current state** is permissible.
  - We **don’t care** if state space is **exponential**, because we don’t solve the problem by dynamic programming.
Dynamic Programming Model

- Formulate problem with **dynamic programming** model.
  - Rather than constraint set.
  - Problem must have **recursive** structure.
  - But there is great **flexibility** to represent constraints and objective function.
  - Any function of **current state** is permissible.
  - We **don’t care** if state space is **exponential**, because we don’t solve the problem by dynamic programming.

- State variables are the same as in relaxed DD.
  - Must also specify **state merger** rule.
  - Much as one must **linearize** IP constraints, or perhaps add valid inequalities.
Dynamic Programming Model

- Max stable set problem on a graph.
  - **State** = set of vertices that can be added to stable set.
  - **State merger** = union

- Max cut problem on a graph.
  - **State** = marginal benefit of placing each remaining vertex on left side of cut.
  - **State merger** =
    - Componentwise min if all components $\geq 0$ or all $\leq 0$; 0 otherwise
    - Adjust incoming arc weights

- Max 2-SAT.
  - Similar to max cut.
Branching Algorithm

- Solve optimization problem using a novel branch-and-bound algorithm.
  - Branch on nodes in last exact layer of relaxed decision diagram.
    - ...rather than branch on variables.
    - Create a new relaxed DD rooted at each branching node.
    - Prune search tree using bounds from relaxed DD.

Bergman, Ciré, van Hoeve, JH (2016)
Branching Algorithm

- Solve optimization problem using a novel branch-and-bound algorithm.
  - Branch on nodes in last exact layer of relaxed decision diagram.
    - …rather than branch on variables.
    - Create a new relaxed DD rooted at each branching node.
    - Prune search tree using bounds from relaxed DD.
  - Advantage: a manageable number of states may be reachable in first few layers.
    - …even if the state space is exponential.
    - Alternative way of dealing with curse of dimensionality.

Bergman, Ciré, van Hoeve, JH (2016)
Branching Algorithm

Branching in a relaxed decision diagram

Diagram is exact down to here
Branching in a relaxed decision diagram

Branch on nodes in this layer
Branching in a relaxed decision diagram

First branch

New relaxed decision diagram
Branching Algorithm

Branching in a relaxed decision diagram

Second branch
Branching Algorithm

Branching in a relaxed decision diagram

Third branch

Continue recursively
State Space Relaxation?

- This is very different from state space relaxation.
  - Problem is not solved by dynamic programming.
  - Relaxation created by merging nodes of DD
    - ...rather than mapping into smaller state space.
  - Relaxation is constructed dynamically
    - ...as relaxed DD is built.
  - Relaxation uses same state variables as exact formulation
    - ...which allows branching in relaxed DD

Christofides, Mingozi, Toth (1981)
Computational performance

• Computational results…
  – Applied to stable set, max cut, max 2-SAT.
    – Superior to commercial MIP solver (CPLEX) on most instances.
    – Obtained best known solution on some max cut instances.
  – Slightly slower than MIP on stable set with precomputed clique cover model, but…

Bergman, Ciré, van Hoeve, JH (2016)
Max cut on a graph

Avg. solution time vs graph density

30 vertices

Computational performance

Average solution time (sec)

Density of graph

- CPLEX
- MDDs
Computational performance

Max 2-SAT

Performance profile

30 variables
Max 2-SAT

Performance profile

40 variables

Computational performance

Number of instances solved vs. Computation time (sec) for MDDs and CPLEX.
Computational performance

• Potential to scale up
  – No need to load large inequality model into solver.
  – **Parallelizes** very effectively
    – **Near-linear** speedup.
    – Much better than mixed integer programming.
Computational performance

- In all computational comparisons so far...
  - Problem is *easily formulated for IP*.
- DD-based optimization is most competitive when...
  - Problem has a recursive dynamic programming model...
  - and *no convenient IP model*.
- Such as...
  - Sequencing and scheduling problems (next talk)
  - DP problems with exponential state space
    - New approach to “curse of dimensionality”
  - Problems with nonconvex, nonseparable objective function…
Modeling the Objective Function

- Weighted DD can represent any objective function
  - Separable functions are the easiest, but any nonseparable function is possible.
  - Can be nonlinear, nonconvex, etc.
  - The issue is complexity of resulting DD
Modeling the Objective Function

• Weighted DD can represent any objective function
  – Separable functions are the easiest, but any nonseparable function is possible.
  – Can be nonlinear, nonconvex, etc.
  – The issue is complexity of resulting DD

• Multiple encodings
  – A given objective function can be encoded by multiple assignments of costs to arcs.
  – There is a unique canonical arc cost assignment.
    – Which can reduce size of exact DD.
  – Design state variables accordingly
Set covering with separable cost function

Easy. Just label arcs with weights.

\[
\begin{array}{cccc}
\text{Set} & i & 1 & 2 & 3 & 4 \\
\hline
A & \bullet & \bullet & & & \\
B & \bullet & & \bullet & & \\
C & & \bullet & \bullet & & \\
D & & & & \bullet & \bullet \\
\hline
\text{Weight} & 3 & 5 & 4 & 6 \\
\end{array}
\]

\( x_i = 1 \) when we select set \( i \)
Nonseparable cost function

Now what?

<table>
<thead>
<tr>
<th>x</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1,0,1)</td>
<td>6</td>
</tr>
<tr>
<td>(0,1,1,0)</td>
<td>7</td>
</tr>
<tr>
<td>(0,1,1,1)</td>
<td>8</td>
</tr>
<tr>
<td>(1,0,1,1)</td>
<td>5</td>
</tr>
<tr>
<td>(1,1,0,0)</td>
<td>6</td>
</tr>
<tr>
<td>(1,1,0,1)</td>
<td>8</td>
</tr>
<tr>
<td>(1,1,1,0)</td>
<td>7</td>
</tr>
<tr>
<td>(1,1,1,1)</td>
<td>9</td>
</tr>
</tbody>
</table>
Modeling the Objective Function

Nonseparable cost function

Put costs on leaves of branching tree.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$f(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1,0,1)</td>
<td>6</td>
</tr>
<tr>
<td>(0,1,1,0)</td>
<td>7</td>
</tr>
<tr>
<td>(0,1,1,1)</td>
<td>8</td>
</tr>
<tr>
<td>(1,0,1,1)</td>
<td>5</td>
</tr>
<tr>
<td>(1,1,0,0)</td>
<td>6</td>
</tr>
<tr>
<td>(1,1,0,1)</td>
<td>8</td>
</tr>
<tr>
<td>(1,1,1,0)</td>
<td>7</td>
</tr>
<tr>
<td>(1,1,1,1)</td>
<td>9</td>
</tr>
</tbody>
</table>
Nonseparable cost function

Put costs on leaves of branching tree.

But now we can’t reduce the tree to an efficient decision diagram.
Nonseparable cost function

Put costs on leaves of branching tree.

But now we can’t reduce the tree to an efficient decision diagram.

We will rearrange costs to obtain canonical costs.
Nonseparable cost function

Put costs on leaves of branching tree.

But now we can’t reduce the tree to an efficient decision diagram.

We will rearrange costs to obtain canonical costs.
Modeling the Objective Function

Nonseparable cost function

Put costs on leaves of branching tree.

But now we can’t reduce the tree to an efficient decision diagram.

We will rearrange costs to obtain canonical costs.
Modeling the Objective Function

Nonseparable cost function

Put costs on leaves of branching tree.

But now we can’t reduce the tree to an efficient decision diagram.

We will rearrange costs to obtain canonical costs.
Nonseparable cost function

Now the tree can be reduced.
Nonseparable cost function

Now the tree can be reduced.
Modeling the Objective Function

Nonseparable cost function

DD is larger than reduced unweighted DD, but still compact.
Modeling the Objective Function

**Theorem.** For a given variable ordering, a given objective function is represented by a unique weighted decision diagram with canonical costs.
Inventory Management Example

- In each period $i$, we have:
  - Demand $d_i$
  - Unit production cost $c_i$
  - Warehouse space $m$
  - Unit holding cost $h_i$
- In each period, we decide:
  - Production level $x_i$
  - Stock level $s_i$
- Objective:
  - Meet demand each period while minimizing production and holding costs.
Reducing the Transition Graph

\[ g_i(s_i) = \min_{x_i} \{ h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i) \} \]

Arcs leaving each node are very similar.
- Transition to the same states.
- Have the same costs, up to an offset.
Inventory Problem

\[ g_i(s_i) = \min_{x_i} \{ h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i) \} \]

To equalize controls, let
\[ x'_i = s_i + x_i - d_i \]

Be the stock level in next period.
Inventory Problem

\[ g_i(s_i) = \min_{x_i} \{ h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i) \} \]

To equalize controls, let

\[ x_i' = s_i + x_i - d_i \]

Be the stock level in next period.
Inventory Problem

New recursion:

\[ g_i(s_i) = \min_{x'_i} \left\{ h_i s_i + c_i (x'_i - s_i + d_i) + g_{i+1}(x'_i) \right\} \]

To equalize controls, let

\[ x'_i = s_i + x_i - d_i \]

Be the stock level in next period.
Inventory Problem

\[ g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i (x_i' - s_i + d_i) + g_{i+1}(x_i') \right\} \]

To obtain canonical costs, subtract \( c_i (m - s_i) + h_i s_i \) from cost on each arc \((s_i, s_{i+1})\).
Inventory Problem

To obtain canonical costs, subtract \( c_i(m - s_i) + h_i s_i \) from cost on each arc \((s_i, s_{i+1})\).

Add these offsets to incoming arcs.

\[
g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}
\]
Inventory Problem

\[ g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i (x_i' - s_i + d_i) + g_{i+1}(x_i') \right\} \]

To obtain canonical costs, subtract \( c_i (m - s_i) + h_i s_i \) from cost on each arc \((s_i, s_{i+1})\).

Add these offsets to incoming arcs.
Inventory Problem

$$g_i(s_i) = \min_{x'_i} \left\{ h_i s_i + c_i (x'_i - s_i + d_i) + g_{i+1}(x'_i) \right\}$$

To obtain canonical costs, subtract $c_i (m - s_i) + h_i s_i$ from cost on each arc $(s_i, s_{i+1})$.

Add these offsets to incoming arcs.

Now outgoing arcs look alike.

And all arcs into state $s_i$ have the same cost

$$\bar{c}_i(s_{i+1}) = s_{i+1} h_{i+1} + c_i (d_i - s_{i+1} - m) + c_{i+1} (m - s_{i+1})$$
Inventory Problem

\[ g_i(s_i) = \min_{x'_i} \left\{ h_i s_i + c_i (x'_i - s_i + d_i) + g_{i+1}(x'_i) \right\} \]

These are canonical costs with offset \( \min_{s_{i+1}} \{ \bar{c}_i(s_{i+1}) \} \)
Inventory Problem

New recursion:

\[
g_i = \min_{x_i'} \left\{ h_{i+1} x_i' + c_i (x_i' - m + d_i) + c_{i+1} (m - x_i') + g_{i+1} \right\}
\]

These are canonical costs with offset \( \min_{s_{i+1}} \{ \bar{c}_i (s_{i+1}) \} \)
Inventory Problem

New recursion:

$$g_i = \min_{x'_i} \left\{ h_{i+1} x'_i + c_i (x'_i - m + d_i) + c_{i+1} (m - x'_i) + g_{i+1} \right\}$$

Now there is only one state per period.
Current Research

- Broader applicability
  - Stochastic dynamic programming
  - Continuous global optimization

- Combination with other techniques
  - Lagrangean relaxation.
  - Column generation
  - Logic-based Benders decomposition
    - Solve separation problem
2006

2007

2008
References

2010

2011

2012
References

2013

2014
References

2014


2015


2016
