Decision Diagrams: Tutorial

John Hooker
Carnegie Mellon University

CP Summer School
Cork, Ireland, June 2016
Decision Diagrams

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

• A new perspective on problem solving
  – Constraint programming
  – Discrete optimization
Elements of a DD-based Solver

• **CP solver**
  – Build on *existing solver*.
  – Use *relaxed* DDs for enhanced *propagation*.
  – Plug in DDs as *additional global constraints*.

• **Discrete optimization solver**
  – Obtain *bounds* from *relaxed* DDs.
  – Use *restricted* DDs for *primal heuristic*.
  – Use *dynamic programming* formulation of problem.
  – **Branch** inside relaxed DD.
Decision Diagrams

• **Advantages for constraint programming:**
  - **Stronger** propagation, filtering.
  - Easily added to existing solver.

• **Advantages for optimization:**
  - 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
• **Relaxed** decision diagrams
  – Relaxation by **node merger**
  – Relaxation by **node splitting**
• **Propagation** in relaxed diagrams
• **Restricted** decision diagrams
• **Dynamic programming** model
• **Branching** in a relaxed DD
• Modeling the **objective function**
  – Inventory management example
• **Nonserial** decision diagrams
• 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 representing any given Boolean function.
  – Once the variable ordering is specified.

  Bryant (1986)

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

\[ x_1 = 0 \quad x_1 = 1 \]
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

\{12345\}

\begin{align*}
    x_1 \\
    x_2 \\
    x_3 \\
    x_4 \\
    x_5
\end{align*}
Exact DD for stable set problem

To build DD, associate state with each node

\[ \begin{align*}
\text{1} & \quad \text{2} & \quad \text{3} \\
\text{4} & \quad \text{5} \\
\{12345\} & \quad \{2345\} & \quad \{34\}
\end{align*} \]
Exact DD for stable set problem

To build DD, associate state with each node

\begin{itemize}
\item $x_1$
\item $x_2$
\item $x_3$
\item $x_4$
\item $x_5$
\end{itemize}
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).
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)
Relaxation by Node Merger

• One way to relax a DD is to **merge nodes** during top-down compilation.
  – Make sure **state** of merged node excludes no feasible solutions.

Hoda, van Hoeve, JH (2010)
To build relaxed DD, merge some additional nodes as we go along.

\{12345\}

\begin{align*}
&x_1 \\
&x_2 \\
&x_3 \\
&x_4 \\
&x_5
\end{align*}
To build relaxed DD, merge some additional nodes as we go along.
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.
To build **relaxed** DD, merge some additional nodes as we go along.

Take the **union** of merged states.
Represents 11 solutions, including 9 feasible solutions

Width = 2
Represents 11 solutions, including 9 feasible solutions.

Width = 2

Longest path (90) gives bound on optimal value (70)
Relaxation by Node Splitting

- Alternate relaxation method: **node refinement** during top-down compilation
  - Start with DD of width 1 representing Cartesian product of variable domains.
  - Split nodes so as to remove some infeasible paths.

---

Andersen, Hadžić, JH, Tiedemann (2007)
Aim for width = 2

Start with DD of width 1

32 solutions, 9 of which are feasible
Start with DD of width 1

Examine states that result from arcs leaving top node.

Aim for width = 2
Aim for width = 2

Start with DD of width 1

Can split states if they are different (they are).
Aim for width = 2

Start with DD of width 1

Can split states if they are different (they are).
Aim for width = 2

Start with DD of width 1

Examine states in next layer.
Aim for width $= 2$

Start with DD of width 1

Examine states in next layer.

All distinct, split arbitrarily.
Aim for width = 2

Start with DD of width 1

Examine states in next layer.

All distinct, split arbitrarily.
Aim for width = 2

Start with DD of width 1

Repeat.

Two states are identical and are not split.
Aim for width = 2

Start with DD of width 1

Repeat.

Two states are identical and are not split.
Aim for width = 2

Start with DD of width 1

Repeat.

Two states are identical and are not split.
Aim for width = 2

Start with DD of width 1

Repeat.

Two states are identical and are not split.
Aim for width = 2

Start with DD of width 1

12 solutions, 9 of which are feasible
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)
Propagation in Relaxed DDs

- Propagate through **relaxed DD** rather than domain store.
  - DD conveys more information.
- This was first application of relaxed DDs.
  - Applied to multiple alldiffs (graph coloring).

Andersen, Hadžić, JH, Tiedemann (2007)
Propagation in Relaxed DDs

• **Example 1: multiple alldiffs**
  – Propagate \texttt{alldiff}(x_1, \ldots, x_4)
  – Through a given DD relaxation.

• **Example 2: single-machine scheduling with time windows.**
  – Propagate alldiff + time windows.
Suppose this is a relaxed DD for the problem.

Indicate multiple arcs with arc domains

Propagate \text{alldiff}(x_1, \ldots, x_4)
alldiff provides no filtering for domain store

\{1234\}

\{12345\}

\{1234\}

\{12345\}

\{12345\}
For purposes of filtering alldiff, introduce state \((A, S)\)

\(A = \{\text{jobs on all paths to node}\}\)

\(S = \{\text{jobs on some path to node}\}\)
For purposes of filtering alldiff, introduce state \((A, S)\)

\[ A = \{\text{jobs on all paths to node}\} \]

\[ S = \{\text{jobs on some path to node}\} \]
For purposes of filtering alldiff, introduce state \((A,S)\)

\[ A = \{ \text{jobs on all paths to node} \} \]

\[ S = \{ \text{jobs on some path to node} \} \]

Can remove 4 from outgoing arc domain
For purposes of filtering alldiff, introduce state \((A, S)\)

\[ A = \{ \text{jobs on all paths to node} \} \]

\[ S = \{ \text{jobs on some path to node} \} \]

Can remove 4 from outgoing arc domain
For purposes of filtering alldiff, introduce state \((A,S)\)

\(A = \{\text{jobs on all paths to node}\}\)

\(S = \{\text{jobs on some path to node}\}\)
Can remove 3 from outgoing arc domain
Can remove 3 from outgoing arc domain
Can remove 5 from outgoing arc domain
Can remove 5 from outgoing arc domain
We have a Hall set.

Can remove 1, 2 from outgoing arc domain.
We have a Hall set.
Can remove 1, 2 from outgoing arc domain
Now some domains can be reduced, resulting in less search.
Now some domains can be reduced, resulting in less search.
Can follow this with a **bottom-up** pass.
Propagation in Relaxed DDs

• **Computational results**
  – Reduced search trees from 1+ million nodes to 1 node.
  – Reduced computation time by one order of magnitude.

Andersen, Hadžić, JH, Tiedemann (2007)
Propagation in Relaxed DDs

• Example 2: single-machine scheduling with time windows.
  – Schedule jobs sequentially, no overlap.
  – Each has given processing time and deadline.
  – Other constraints.
  – $x_i = i$ th job in sequence

• Use same relaxed DD as before.
  – Suppose we have already propagated $\text{alldiff}(x_1, \ldots, x_n)$.
  – Now propagate $\text{time windows}$. 
Current relaxed DD

\[
\begin{align*}
\{1234\} & \quad \{12\} & \quad \{3\} & \quad \{4\} \\
\{1235\} & \quad \{12\} & \quad \{3\} & \quad \{1\} & \quad \{5\} \\
\{123\} & \quad \{3\} & \quad \{1\} & \quad \{2\} & \quad \{1\} \\
\{245\} & \quad \{45\} & \quad \{2\} & \quad \{1\} \\
\end{align*}
\]
For purposes of propagating deadlines, let $\text{state} = \min \text{ latest finish time}$

<table>
<thead>
<tr>
<th>job</th>
<th>Window</th>
<th>Proc time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,4]</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>[3,7]</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>[1,8]</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>[5,7]</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>[2,10]</td>
<td>3</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For purposes of propagating deadlines, let \textbf{state} = min latest finish time

<table>
<thead>
<tr>
<th>job</th>
<th>Window</th>
<th>Proc time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,4]</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>[3,7]</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>[1,8]</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>[5,7]</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>[2,10]</td>
<td>3</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For purposes of propagating deadlines, let \textbf{state} = \text{min latest finish time}

**Table:**

<table>
<thead>
<tr>
<th>job</th>
<th>Window</th>
<th>Proc time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,4]</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>[3,7]</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>[1,8]</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>[5,7]</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>[2,10]</td>
<td>3</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram Notes:**

- Can be deleted, because jobs must be late.
For purposes of propagating deadlines, let \textbf{state} = \text{min latest finish time}

<table>
<thead>
<tr>
<th>job</th>
<th>Window</th>
<th>Proc time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,4]</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>[3,7]</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>[1,8]</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>[5,7]</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>[2,10]</td>
<td>3</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For purposes of propagating deadlines, let \textbf{state} = min latest finish time.

<table>
<thead>
<tr>
<th>job</th>
<th>Window</th>
<th>Proc time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,4]</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>[3,7]</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>[1,8]</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>[5,7]</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>[2,10]</td>
<td>3</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Job 4 can be deleted.
For purposes of propagating deadlines, let $\text{state} = \min\text{ latest finish time}$.

<table>
<thead>
<tr>
<th>job</th>
<th>Window</th>
<th>Proc time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0,4]</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>[3,7]</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>[1,8]</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>[5,7]</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>[2,10]</td>
<td>3</td>
</tr>
</tbody>
</table>

etc.
Domains can be reduced further

{1234}
{1235}
{123}
{245}

{12}
{3}
{1}
{5}

x_1
x_2
x_3
x_4
x_5
Domains can be reduced further.
**CP Solver**

- **Enhance** existing solver with DD-based *propagation*.
  - DD serves as *enhanced* domain store.
  - Can use **one or more** DDs.
    - Different *subsets of variables*
    - Different *variable orderings*
    - Propagate each constraint through suitable DD(s).
  - **Plug in** each DD as a new global constraint.

*Ciré, van Hoeve (2013)*
• Computational results.
  – Traveling salesman problem with time windows.
    • That is, single-machine scheduling with time windows and sequence-dependent setup times.
  – Dumas/Anscheuer instances.

Ciré, van Hoeve (2013)
CPO = CP Optimizer

Computation time scatter plot, lex search

- Pure CP better
- CP + DD better
CPO = CP Optimizer

Computation time scatter plot, depth-first search

Pure CP better

CP + DD better
CPO = CP Optimizer

Performance profile, depth-first search

Number of instances solved

Time(s)

CPO
CPO+MDD - width 1024
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. \( x_1, x_2 \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>●</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Restricted DD of width 4

Several shortest paths have length 2.

All are minimum covers.

41 paths (< 52 feasible solutions)
Restricted DD of width 4

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 optimality gap for set covering, $n$ variables. The graph compares Restricted DDs and Primal heuristic at the root node of CPLEX. The x-axis represents $n$ variables, and the y-axis represents the average optimality gap (%). The graph shows a clear distinction between Restricted DDs (DD) and the Primal heuristic (IP), with the optimality gap increasing as $n$ increases.]
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.
Dynamic Programming Model

• Max stable set problem on a graph.
  – **State** = set of vertices that can be added to stable set.

Recursion:

\[
g(J) = \max_{j \in J} \left\{ w_j + g(J \setminus N(j)) \right\}
\]

Cost-to-go

State

Immediate cost (edge weight)

Vertex \( j \) and neighbors

Boundary condition:

\[ g(\emptyset) = 0 \]

Optimal value:

\[ g(\{1, \ldots, n\}) \]
Dynamic Programming Model

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

Recursion:

\[
g(J) = \max_{j \in J} \left\{ w_j + g(J \setminus N(j)) \right\}
\]

Merger of states in \( M \) = \bigcup_{J \in M} J

Boundary condition:

\[
g(\emptyset) = 0
\]

Optimal value:

\[
g(\{1, \ldots, n\})
\]
Dynamic Programming Model

- **Single-machine scheduling with due dates**
  - Minimize total tardiness.
  - **State** = (set of jobs not yet processed, latest finish time of jobs processed so far)

\[
g_i(J_i, f_i) = \max_{j \in J_i} \left\{ (f_i + p_j - d_j)^+ + g_{i+1}(J_i \setminus \{j\}, f_i + p_j) \right\}
\]

Boundary condition:
\[g_{n+1}(\emptyset, f_{n+1}) = 0\]

Optimal value:
\[g_1(\{1, \ldots, n\}, 0)\]
Dynamic Programming Model

- Single-machine scheduling with due dates
  - Minimize total tardiness.
  - State = (set of jobs not yet processed, latest finish time of jobs processed so far)
  - State merger = union, min

\[ g_i(J_i, f_i) = \max_{j \in J_i} \left\{ (f_i + p_j - d_j)^+ + g_{i+1}(J_i \setminus \{j\}, f_i + p_j) \right\} \]

Cost-to-go
Jobs remaining
Last finish time
Tardiness of job j

Boundary condition:
\[ g_{n+1}(\emptyset, f_{n+1}) = 0 \]
Optimal value:
\[ g_1(\{1, \ldots, n\}, 0) \]

Merger of states in \( M \):
\[ \left( \bigcup_{(J_i, f_i) \in M} J_i, \min_{(J_i, f_i) \in M} \{f_i\} \right) \]
Dynamic Programming Model

- Single machine scheduling with due dates
  - Easy to add constraints that are functions of current state
    - Release times
    - Shutdown periods
    - Precedence constraints on jobs
  - Easy to use more complicated cost function that is a function of current state
    - Step functions, etc.
    - Cost that depends on which jobs have been processed.
Dynamic Programming Model

- **Scheduling with sequence-dependent setup times**
  - **State** = \((J_i, \text{last job processed, } f_i)\)
  - **State merger** requires modification of states

\[ g_i(J_i, \ell_i, f_i) = \max_{j \in J_i} \left\{ (f_i + p_{\ell_ij} - d_j)^+ + g_{i+1}(J_i \setminus \{j\}, j, f_i + p_{\ell_ij}) \right\} \]

- Last job processed
- Tardiness of job \(j\)
- Processing + setup time
Dynamic Programming Model

- **Scheduling with sequence-dependent setup times**
  - To allow for **state merger**:
  - **State** = \( (J_i, \text{set } L_i \text{ of pairs } (\ell_i, f_i), \text{representing jobs that could have been the last processed}) \)

\[
g_i(J_i, L_i) = \max_{j \in J_i} \left\{ \left( \min_{(\ell_i, f_i) \in L_i} \{ f_i + p_{\ell_i j} \} - d_j \right)^+ \right. \\
+ g_{i+1} \left( J_i \setminus \{ j \}, \left\{ \left( j, \min_{(\ell_i, f_i) \in L_i} \{ f_i + p_{\ell_i j} \} \right) \right\} \right) \right\}
\]

**Merger of states in** \( M \) = \( \bigcup_{(J_i, L_i) \in M} J_i, \bigcup_{(J_i, L_i) \in M} L_i, \) \( (J_i, L_i) \in M \)
Dynamic Programming Model

- **Max cut problem on a graph.**
  - Partition nodes into 2 sets so as to maximize total weight of connecting edges.
  - **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 \textbf{branch-and-bound} algorithm.
  – Branch on nodes in \textbf{last exact layer} of relaxed decision diagram.
    – …rather than branch on variables.
    – Create a new \textbf{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 \textbf{branch-and-bound} algorithm.
  – Branch on nodes in \textbf{last exact layer} of relaxed decision diagram.
    – …rather than branch on variables.
    – Create a new \textbf{relaxed DD rooted} at each branching node.
    – Prune search tree using bounds from relaxed DD.
  – Advantage: a manageable number states may be reachable in first few layers.
    – …even if the state space is \textbf{exponential}.
    – Alternative way of dealing with \textbf{curse of dimensionality}.

Bergman, Ciré, van Hoeve, JH (2016)
Branching in a relaxed decision diagram

Diagram is exact down to here
Branching Algorithm

Branching in a relaxed decision diagram

Branch on nodes in this layer
Branching Algorithm

Branching in a relaxed decision diagram

First branch

New relaxed decision diagram
Branching in a relaxed decision diagram

**Branching Algorithm**

Prune this branch if **cost bound** from relaxed DD is **no better** than cost of best feasible solution found so far (branch and bound).
Branching in a relaxed decision diagram

Prune this branch if cost bound from relaxed DD is no better than cost of best feasible solution found so far (branch and bound).
Branching Algorithm

Branching in a relaxed decision diagram

Prune this branch if cost bound from relaxed DD is no better than cost of best feasible solution found so far (branch and bound).
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)
Discrete Optimization Solver

• Enhance existing solver with DDs
  – Better **bounds** from **relaxed** DDs.
  – Better **primal heuristic** using **restricted** DDs.
  – Add to existing LP relaxation and primal heuristics.

• Use stand-alone DD-based solver
  – Obtain **bounds** from **relaxed** DDs.
  – Use **restricted** DDs for **primal heuristic**.
  – Use **dynamic programming** formulation of problem.
  – **Branch** inside relaxed DD.
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)
Computational performance

Max cut on a graph

Avg. solution time vs graph density

30 vertices
Computational performance

Max 2-SAT

Performance profile

30 variables

![Graph showing computational performance](image)
Computational performance

Max 2-SAT

Performance profile

40 variables

![Graph showing computational performance comparison between MDDs and CPLEX. The x-axis represents computation time (sec) ranging from 0.1 to 1000, and the y-axis represents the number of instances solved, ranging from 0 to 100. Two lines are plotted: blue for MDDs and orange for 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.
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
  – 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
Modeling the Objective Function

Set covering with separable cost function

Easy. Just label arcs with weights.

\[
x_i = 1 \text{ when we select set } i
\]

<table>
<thead>
<tr>
<th>Set ( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>B</td>
<td>●</td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>C</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>●</td>
<td></td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Weight</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Graph representing the sets and weights:

- Set A covers nodes 1 and 2.
- Set B covers nodes 2 and 3.
- Set C covers nodes 3 and 4.
- Set D covers nodes 4 and 5.

Weights at nodes:
- Node 1: 1
- Node 2: 3
- Node 3: 4
- Node 4: 4
- Node 5: 5
- Node 6: 6
Modeling the Objective Function

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>
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.
Modeling the Objective Function

Nonseparable cost function

Now the tree can be reduced.
Nonseparable cost function

Now the tree can be reduced.
Nonseparable cost function

DD is larger than reduced unweighted DD, but still compact.
Theorem. For a given variable ordering, a given objective function is represented by a unique weighted decision diagram with canonical costs.

JH (2013),
Similar result for AADDs:
Sanner & McAllester (2005)
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

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

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

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

To equalize controls, let
\[ x'_i = s_i + x_i - d_i \]
be the stock level in next period.
Inventory Problem

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

To equalize controls, let

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

Be the stock level in next period.
Inventory Problem

New recursion:

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

To equalize controls, let

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

Be the stock level in next period.
To obtain canonical costs, subtract from cost on each arc \((s_i, s_{i+1})\).
Inventory Problem

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

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.
Inventory Problem

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

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 $$\min_{s_{i+1}} \left\{ \tilde{c}_i(s_{i+1}) \right\}$$
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 \[\min_{s_{i+1}} \left\{ \bar{c}_i(s_{i+1}) \right\} \]
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.

JH (2013)
Nonserial Decision Diagrams

• Analogous to nonserial dynamic programming, independently(?) rediscovered many times:
  – Nonserial DP (1972)
  – Constraint satisfaction (1981)
  – Data base queries (1983)
  – $k$-trees (1985)
  – Belief logics (1986)
  – Bucket elimination (1987)
  – Bayesian networks (1988)
  – Pseudoboollean optimization (1990)
  – Location analysis (1994)
Set Partitioning example

Find collection of sets that partition elements A, B, C, D

<table>
<thead>
<tr>
<th>Sets</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>
Set Partitioning example

Find collection of sets that partition elements A, B, C, D

Sets

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>

For example…
Set Partitioning example

Find collection of sets that partition elements A, B, C, D

<table>
<thead>
<tr>
<th>Sets</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

Or…
Set Partitioning example

Find collection of sets that partition elements A, B, C, D

Sets

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0-1 formulation

\[
\begin{align*}
    x_1 + x_2 + x_3 &= 1 \\
    x_2 + x_4 &= 1 \\
    x_3 + x_5 + x_6 &= 1 \\
    x_4 + x_6 &= 1
\end{align*}
\]

\[x_j = 1 \implies \text{set } j \text{ selected}\]
Set Partitioning example

Dependency graph

0-1 formulation

\[
\begin{align*}
    x_1 + x_2 + x_3 &= 1 \\
    x_2 + x_4 &= 1 \\
    x_3 + x_5 + x_6 &= 1 \\
    x_4 + x_6 &= 1
\end{align*}
\]

\[x_j = 1 \quad \Rightarrow \quad \text{set } j \text{ selected}\]
Set Partitioning example

Dependency graph

x₁ ← x₃ → x₅

x₂ → x₄ → x₆

Enumeration order

x₂
x₃
x₄
x₁
x₅
x₆
Set Partitioning example

Dependency graph

Enumeration order

x_1 \leftarrow x_3 \rightarrow x_5

x_2 \leftarrow x_4 \rightarrow x_6

x_2 \rightarrow x_3

x_3 \rightarrow x_4

x_4 \rightarrow x_5

x_5 \rightarrow x_6
Set Partitioning example

Dependency graph

X1 ← X3 → X5
X2 ← X4 → X6

Enumeration order

x2
x3
x4
x5
x6
Set Partitioning example

Dependency graph

Enumeration order

$x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$
Set Partitioning example

Dependency graph

Enumeration order

$x_1 \leftarrow x_3 \rightarrow x_5$

$x_2 \rightarrow x_4 \rightarrow x_6$

$x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_1$

$x_5 \rightarrow x_6$

$x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_6 \rightarrow x_1$
Set Partitioning example

Dependency graph

Enumeration order
Set Partitioning example

Dependency graph

Enumeration order
Set Partitioning example

Dependency graph

Induced width = 3
(max in-degree)

Enumeration order
Set Partitioning example

Solution by nonserial DP

Enumeration order
Set Partitioning example

Solution by nonserial DP

<table>
<thead>
<tr>
<th>Sets</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

Diagram:

- $x_2$
- $x_2x_3$
- $x_3x_4$
- $x_1$
- $x_3x_4x_5$
- $x_6$
Set Partitioning example

Feasible solution

<table>
<thead>
<tr>
<th>x_2</th>
<th>x_2x_3</th>
<th>x_3x_4</th>
<th>x_1</th>
<th>x_3x_4x_5</th>
<th>x_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Sets

<table>
<thead>
<tr>
<th>Sets</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>
Set Partitioning example

Feasible solution

\[
\begin{align*}
x_2 & \quad x_3 \\
x_2x_3 & \quad x_3x_4 \\
x_3x_4x_5 & \\
x_6
\end{align*}
\]

Sets

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{pmatrix}
1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0
\end{pmatrix}
\]
Set Partitioning example

Feasible solution

Sets

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x_2</th>
<th>x_2 \cdot x_3</th>
<th>x_3 \cdot x_4</th>
<th>x_1</th>
<th>x_3 \cdot x_4 \cdot x_5</th>
<th>x_6</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>01</td>
<td>00</td>
<td>11</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>00</td>
<td>11</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>00</td>
<td>11</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>00</td>
<td>11</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>00</td>
<td>11</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>00</td>
<td>11</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Set Partitioning example

Solution by nonserial DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>00</th>
<th>01</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_2$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$x_2x_3$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$x_3x_4$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$x_1$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$x_3x_4x_5$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$x_6$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>

Serialized DP

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>00</th>
<th>01</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_2$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$x_2x_3$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$x_3x_3x_4$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$x_1x_2x_3x_4$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$x_3x_4x_5$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>$x_6$</td>
<td></td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
</tbody>
</table>
Set Partitioning example
BDD vs. DP Solution

BDD

x_2
x_2 x_3
x_3 x_4
x_1
x_3 x_4 x_5
x_6

serialized DP

x_2
x_2 x_3
x_3 x_3 x_4
x_1 x_2 x_3 x_4
x_3 x_4 x_5
x_6
BDD vs. DP Solution

BDD

```
x_2
x_2 x_3
x_3 x_4
x_1
x_3 x_4 x_5
x_6
```

```
0 1
0 1
0 1
1 0
0 0
1 1
0 0
0 0
```

Serialized DP

```
x_2
x_2 x_3
x_3 x_3 x_4
x_1 x_2 x_3 x_4
x_3 x_4 x_5
x_6
```

```
0 1
0 1
0 1
1 0
0 0
1 1
0 0
0 0
```

Deleted
BDD vs. DP Solution

BDD

x₂

x₂x₃

x₃x₄

x₁

x₃x₄x₅

x₆

0 1

0 1

00 01 10

001 011 100

1001 0011 0100

011 000

110

0 1

x₂x₃

x₃x₃x₄

x₁x₂x₃x₄

x₃x₄x₅

x₆

0 1

00 01 10

001 011 100

1001 0011 0100

010 011 110

0 1

Merged
Set Partitioning example

Solution by nonserial DP

$X_2$

$X_2X_3$

$X_3X_4$

$X_1$

$X_3X_4X_5$

$X_6$
Set Partitioning example

Solution by nonserial DP

Nonserial BDD
Constructing the Join Tree

$x_2 x_3 x_4 x_1 x_5 x_6$

Clique in the dependency graph

$x_1 x_2 x_3$

$x_1 \leftrightarrow x_2 \leftrightarrow x_3 \leftrightarrow x_4 \leftrightarrow x_5 \leftrightarrow x_6$
Constructing the Join Tree

\[ x_2 \ x_3 \ x_4 \ x_1 \ x_5 \ x_6 \]

Clique in the dependency graph

\[ x_1x_2x_3 \]

\[ x_2x_3x_4 \]
Constructing the Join Tree

Clique in the dependency graph

\[ x_2 \ x_3 \ x_4 \ x_1 \ x_5 \ x_6 \]

\[ x_1 x_2 x_3 \]

\[ x_2 x_3 x_4 \]

\[ x_3 x_4 x_5 x_6 \]
Constructing the Join Tree

$x_2 \ x_3 \ x_4 \ x_1 \ x_5 \ x_6$

Dependency graph

Join graph

Connect nodes with common variables
Constructing the Join Tree

\[ x_2 \ x_3 \ x_4 \ x_1 \ x_5 \ x_6 \]

Join graph

Dependency graph

\[ x_j \text{ occurs along every path connecting } x_j \text{ with } x_j \]
Constructing the Join Tree

This can be viewed as the **constraint dual**

Binary constraints equate common variables in subproblems
Constructing the Join Tree

\[ x_2 \ x_3 \ x_4 \ x_1 \ x_5 \ x_6 \]

Dependency graph

Join graph

Some edges may be redundant when equating variables
Constructing the Join Tree

$X_2 \ X_3 \ X_4 \ X_1 \ X_5 \ X_6$

Dependency graph

Join tree

Removing redundant edges yields join tree
Constructing the Join Tree

Dependency graph

$X_1 \leftarrow X_2 \rightarrow X_3 \leftarrow X_4 \rightarrow X_5 \leftarrow X_6$

Join tree

$X_1X_2X_3 \rightarrow \begin{array}{c} X_2X_3 \\ X_2X_3X_4 \\ X_3X_4 \\ X_3X_4X_5X_6 \end{array}$

Max node cardinality is tree width $+ 1 = 3 + 1$
Constructing the Join Tree

\[ x_2 \ x_3 \ x_4 \ x_1 \ x_5 \ x_6 \]

Dependency graph

Join tree

Induced width = tree width = 3
Designing the Nonserial BDD

BDD design

\[ x_2 \]

Join tree

\[ x_2 \rightarrow x_2 x_3 x_1 \]
\[ x_2 x_3 \rightarrow x_2 x_3 x_4 \]
\[ x_3 x_4 \rightarrow x_3 x_4 x_5 x_6 \]
Designing the Nonserial BDD

BDD design

$\bar{x_2} \cdot x_3 \cdot x_4 \cdot \bar{x_1} \cdot x_5 \cdot x_6$

Join tree

$x_2 \cdot x_3 \cdot x_1$

$x_2 \cdot x_3 \cdot x_4$

$x_3 \cdot x_4$

$x_3 \cdot x_4 \cdot x_5 \cdot x_6$
Designing the Nonserial BDD

x_2 \ x_3 \ x_4 \ x_1 \ x_5 \ x_6

BDD design

Join tree

x_2 x_3 x_1

x_2 x_3

x_2 x_3 x_4

x_3 x_4

x_3 x_4 x_5 x_6
Designing the Nonserial BDD

\[
x_2 \ x_3 \ x_4 \ x_1 \ x_5 \ x_6
\]

BDD design

Join tree
Designing the Nonserial BDD

BDD design

Join tree
Designing the Nonserial BDD

BDD design

Join tree

x_2 x_3 x_4 x_1 x_5 x_6

x_3 x_4 x_5 x_6

x_2 x_3 x_4

x_3 x_4

x_2 x_3 x_1

x_2 x_3
Designing the Nonserial BDD

BDD design

Nonserial BDD
Another Variable Ordering

\[ x_3 \ x_6 \ x_2 \ x_5 \ x_1 \ x_4 \]

Dependency graph

Join graph

Induced width = 2
Constructing the Join Tree

$x_3 \ x_6 \ x_2 \ x_5 \ x_1 \ x_4$

Dependency graph

Join tree

Induced width = 2

Tree width = 2
Designing the BDD

BDD design

Join tree

Tree width = 2
Designing the BDD

$x_3 \ x_6 \ x_2 \ x_5 \ x_1 \ x_4$

BDD design

$x_3$

$x_3x_6$

Join tree

$x_3x_2x_1$

$x_3x_6x_5$

$x_3x_2$

$x_3x_6$

$x_3x_6x_2$

$x_6x_2$

$x_6x_2x_4$

Tree width = 2
Designing the BDD

\[ x_3 \ x_6 \ x_2 \ x_5 \ x_1 \ x_4 \]

BDD design

Join tree

Tree width = 2
Designing the BDD

\[ x_3 \ x_6 \ x_2 \ x_5 \ x_1 \ x_4 \]

**BDD design**

**Join tree**

Tree width = 2
Nonserial BDD

\[
x_3 \ x_6 \ x_2 \ x_5 \ x_1 \ x_4
\]
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
References

2006


2007


2008

References

2010

2011

2012
References

2013


2014

References

2014

2015

2016