The Separation Problem for Binary Decision Diagrams

J. N. Hooker

Joint work with André Ciré

Carnegie Mellon University

ISAIM 2014
Separation Problem in Optimization

• Given a relaxation of an optimization problem…
• Find a constraint that **separates** solution of the relaxation from the feasible set
Separation Problem in Optimization

• Given a relaxation of an optimization problem…
• Find a constraint that **separates** solution of the relaxation from the feasible set
Separation Problem in Optimization

- Now **strengthen** the relaxation with the separating cut.
  - Cuts are usually linear inequalities.
Separation Problem in Optimization

- Now **strengthen** the relaxation with the separating cut.
  - Cuts are usually linear inequalities.
  - Re-solve relaxation and repeat.
Separation Problem in Optimization

• Separation is a **workhorse** in integer and nonlinear programming.
  - Relaxations are usually polyhedral.
  - *Typical problem:* in a **large** known family of cuts, find one that separates.
  - Cannot publish a cutting plane paper without a separation algorithm.
Separation Problem in Optimization

• **Example:** Integer programming
  – **Gomory** cuts
  – **Mixed integer rounding** cuts
  – Separating **knapsack** cuts
  – Separating **cover** inequalities
  – Separating cuts in special families
    • Subtour elimination, combs for TSP
    • Separating flow cuts for fixed-charge network flow
    • etc. (**huge** literature)
Separation Problem for BDDs

- Is there a role for separation in **discrete** relaxations?
- We will look at separation for **binary decision diagrams** (BDDs).
  - Recently used for **discrete optimization** and **constraint programming**.
Separation Problem for BDDs

• Is there a role for separation in discrete relaxations?
• We will look at separation for binary decision diagrams (BDDs).
  – Recently used for discrete optimization and constraint programming.
• Key idea: A relaxed BDD provides a discrete relaxation of the problem as a.
  – How to separate a solution (or family of solutions) from the relaxed BDD?
  – We will focus on separating Benders cuts.
Decision Diagrams

- **Binary decision diagrams (BDDs)** historically used for circuit design and verification.
Decision Diagrams

• **Binary decision diagrams (BDDs)** historically used for circuit design and verification.

• **Compact** graphical representation of **boolean** function.
  – Can also represent **feasible set** of problem with binary variables.
  – Easy generalization to **multivalued** decision diagrams (MDDs) for finite domain variables.
Decision Diagrams

- **Binary decision diagrams (BDDs)** historically used for circuit design and verification.
- **Compact** graphical representation of **boolean** function.
  - Can also represent **feasible set** of problem with binary variables.
  - Easy generalization to **multivalued** decision diagrams (MDDs) for finite domain variables.
- BDD is result of superimposing isomorphic subtrees in a search tree.
  - Unique **reduced** BDD for given variable ordering.
  - A type of “caching.”
Binary Decision Diagrams

- BDD can grow exponentially with problem size.
  - So we use a smaller, **relaxed** BDD that represents **superset** of feasible set.
    - For alldiff systems, reduced search tree from >1 million nodes to 1 node.
    - Subsequent papers with Hadzic, Hoda, van Hoeve, O’Sullivan.

- Example: **independent set problem** on a graph…
Independent Set Problem

Let each vertex have weight $w_i$

Select nonadjacent vertices to maximize $\sum_i w_i x_i$

$x_i = 1$ if vertex $i$ selected
Exact BDD for independent set problem

“zero-suppressed” BDD
Exact BDD for independent set problem

“zero-suppressed” BDD

$x_1 = 1$

$x_2 = 0$
Paths from top to bottom correspond to the 11 feasible solutions.
Paths from top to bottom correspond to the 11 feasible solutions.
Paths from top to bottom correspond to the 11 feasible solutions.
Paths from top to bottom correspond to the 11 feasible solutions.
Paths from top to bottom correspond to the 11 feasible solutions...

...and so forth
For objective function, associate weights with arcs.
For objective function, associate weights with arcs.

Optimal solution is **longest path**.
Objective Function

• In general, objective function can be any *separable function*.
  – Linear or nonlinear, convex or nonconvex

• BDDs can be generalized to *nonseparable* objective functions.
  – There is a unique reduced BDD with *canonical* edge costs.
To build BDD, associate state with each node
To build BDD, associate state with each node.

\{123456\}

\begin{align*}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  x_5 \\
  x_6
\end{align*}
To build BDD, associate state with each node
To build BDD, associate **state** with each node.
Merge nodes that correspond to the same state.
Merge nodes that correspond to the same state.
Merge nodes that correspond to the same state
Width = 2

Merge nodes that correspond to the same state

{x_1} 
{x_2} 
{x_3} 
{x_4} 
{x_5} 
{x_6}
Relaxation Bounding

• To obtain a bound on the objective function:
  – Use a relaxed decision diagram
  – Analogous to linear programming relaxation in MIP
  – This relaxation is discrete.
  – Doesn’t require the linear inequality formulation of MIP.
To build relaxed BDD, merge some additional nodes as we go along.
To build relaxed BDD, merge some additional nodes as we go along
To build **relaxed** BDD, merge some additional nodes as we go along.
To build relaxed BDD, merge some additional nodes as we go along.
To build relaxed BDD, merge some additional nodes as we go along.
To build **relaxed** BDD, merge some additional nodes as we go along.
To build **relaxed** BDD, merge some additional nodes as we go along.
To build relaxed BDD, merge some additional nodes as we go along.
To build relaxed BDD, merge some additional nodes as we go along.
Width = 1

Represents 18 solutions, including 11 feasible solutions

\{123456\}  \{23456\}  \{3456\}  \{456\}  \{56\}  \{6\}  \emptyset

\begin{align*}
1 & \rightarrow 2 \\
1 & \rightarrow 4 \\
1 & \rightarrow 6 \\
2 & \rightarrow 3 \\
2 & \rightarrow 5 \\
3 & \rightarrow 4 \\
3 & \rightarrow 6 \\
4 & \rightarrow 5 \\
5 & \rightarrow 6 \\
\end{align*}
Width = 1

Longest path gives bound of 3 on optimal value of 2
Wider BDDs yield tighter bounds.
  - But take longer to build.
Propagation

• We can propagate by removing arcs from the decision diagram.
  – Rather than removing elements from variable domains.
  – More effective than traditional domain filtering.
  – More information propagated from one constraint to the next.
Suppose this is the relaxed decision diagram
Suppose this is the relaxed decision diagram

Suppose other constraints remove 6 from domain of $x_1$
Suppose this is the relaxed decision diagram

This propagates through the states and removes some arcs.
Suppose this is the relaxed decision diagram

This propagates through the states and removes some arcs.
Separation Problem for BDDs

- Exclude a given partial assignment $x_i = \bar{x}_i$ for $i \in I$.
  - That is, remove all paths in which $x_i = \bar{x}_i$ for $i \in I$.
- Example…
1-arcs from state 1 nodes preserve state 0-arcs switch state to 0.

Original BDD

Remove partial assignment 
\((x_2, x_4) = (1,1)\)
arcs from state 0

Original BDD

Separating BDD

Remove partial assignment

\((x_2, x_4) = (1,1)\)
In principle, a partial assignment can be separated by conjoining two BDDs. However, this introduces an unnecessary data structure.

Separation Algorithm

- Original BDD

\[ \wedge \]

- Width-2 BDD representing negation of partial assignment

- However, this introduces an unnecessary data structure.
Separation Algorithm

• We will propose an algorithm specifically for BDD separation.
  – Exposes essential logic of separation.
  – Operates on original data structure.
  – Allows proof of tighter bounds on growth of the separating BDD as cuts are added.
A node has **state 1** when all incoming paths are excluded.

Otherwise **state 0**.

Assign state 1 to root node.
Duplicate arcs leaving $r$ in original BDD.

Child nodes inherit state of parent node.

$x_1$ unrestricted
$x_1$ unrestricted

$x_2 \neq 1$

1-arcs from state 1 nodes preserve state 1
Original BDD

Separating BDD

\(x_1\) unrestricted

\(x_2 \neq 1\)

1-arcs from state 1 nodes preserve state 1

0-arcs from state 1 nodes
switch to state 0
Duplicate arcs in original BDD.

Child nodes inherit state of parent node.
Duplicate arcs from nodes with state 0, preserving state.
$x_1$ unrestricted

$x_2 \neq 1$

$x_3$ unrestricted

$x_4 \neq 1$

1-arcs from state 1 nodes preserve state
0-arcs switch state to 0.
arcs from state 1 nodes preserve state 0 - arcs switch state to 0.

Original BDD

Separating BDD

$x_1$ unrestricted

$x_2 \neq 1$

$x_3$ unrestricted

$x_4 \neq 1$

$x_5$ unrestricted

Terminate paths at nodes with state 1
Size of Separating BDD

- We wish to separate from a given BDD all solutions $x$ in which $x_i = \bar{x}_i$ for $i \in I$.

**Theorem** (obvious). The separating BDD is at most twice as large as the original BDD.

If only one solution is separated, the separating BDD has at most one additional node per layer.

- This refers to separating BDD created by the algorithm
  - Not necessarily a **reduced** (minimal) BDD.
Size of Separating BDD

- We wish to separate from a given BDD all solutions \( x \) in which \( x_i = \overline{x}_i \) for \( i \in I \).
- Let \( n_i \) be size of layer \( i \) of original BDD.
- Let \( j, k \) be smallest, largest indices in \( I \).

**Theorem** (not so obvious).

Size of layer \( i \) of separating BDD \( \leq \begin{cases} n_i + \varphi_i & \text{if } j \leq i \leq k \\ n_i & \text{otherwise} \end{cases} \)

where \( \varphi_i = \begin{cases} \min\{n_i, \varphi_{i-1}\} & \text{if } i - 1 \in I \\ \min\{n_i, 2\varphi_{i-1}\} & \text{otherwise} \end{cases} \)
Size of Separating BDD

- We wish to separate from a given BDD all solutions \( x \) in which \( x_i = \bar{x}_i \) for \( i \in I \).
- Let \( n_i \) be size of layer \( i \) of original BDD.
- Let \( j,k \) be smallest, largest indices in \( I \).

**Corollary**  Portion of BDD outside the range of indices in \( I \) is unaffected by separation.

- This will be useful in decomposition methods.
Reduced Separating BDD

• Separating BDD generated by the algorithm need not be reduced.
  – The reduced BDD for a Boolean function is the smallest BDD that represents the function.
  – It is unique.

• For example...
Original BDD
BDD that separates $x_2 = 1$ as generated by algorithm

Original BDD

BDD
Original BDD

BDD that separates $x_2 = 1$ as generated by algorithm

Reduced form of separating BDD

$\text{BDD that separates } x_2 = 1 \text{ as generated by algorithm}$
Growth of BDD

- Key question: How fast does the separating BDD grow when a sequence of partial solutions are separated?
  - Traditional relaxation grows **linearly**.
  - One inequality constraint added per solution separated.
  - Will the separating BDD grow linearly?
Worst-Case Growth

• Can **reduced** separating BDD grow exponentially?
  – Yes

• **Example**
  – Start with BDD that represents all Boolean vectors (width 1).
  – Separate:  \((1,*,*,\ldots,*,*,1)\)
    \((*,1,*,\ldots,*,1,*)\)
    \((*,*,1,\ldots,1,*,*)\)
    etc.
Reduced BDD for \( n = 6 \) variables.

It has width \( 2^{n/2} \)
Empirical Growth

• How fast does the separating BDD grow in a realistic optimization algorithm?
  – We will look at a logic-based Benders algorithm
  – …for the home health care delivery problem
Logic-based Benders decomposition is a generalization of classical Benders.

- Address a simplified problem:
  \[ \min f(x) \]
  \[ (x, y) \in S \]

- Benders cut excludes \( \bar{x} \) (and perhaps similar solutions) if it is infeasible in the subproblem.

- Algorithm terminates when \( \bar{x} \) is feasible in the subproblem.
Benders Decomposition

• Logic-based Benders decomposition is a generalization of classical Benders.
  – Master problem is initially a relaxation of the original problem over $x$ (warm start).
  – Relaxation becomes tighter as Benders cuts are added.

Master Problem
Optimize over $x$ subject to Benders cuts

Subproblem
$(\bar{x}, y) \in S$
Using BDDs in Benders:

- We will use a relaxed BDD as initial master problem (warm start).
- Benders cuts exclude partial assignments.
- Add a Benders cut by creating a separating BDD.
Home Health Care

• Home health care delivery problem.
  – Assign nurses to homebound patients.
  – …subject to constraints on nurse qualifications.
  – Route each nurse through assigned patients, observing time windows.
  – Nurse must take a break if day is long enough.
  – Minimize cost in some sense.
Solve with Benders decomposition.
- Assignment problem in master.
- Subproblem generates Benders cuts when there is no feasible schedule for one or more nurses.
- Each cut excludes a partial assignment of nurses to patients.
Home Health Care

- Solve with Benders decomposition.
  - Assignment problem in master.
  - Subproblem generates Benders cuts when there is no feasible schedule for one or more nurses.
  - Each cut excludes a partial assignment of nurses to patients.
- How fast does the separating BDD grow in the master problem?
Results for 20 instances

<table>
<thead>
<tr>
<th>Instance</th>
<th>Iterations</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>set1-n30r0</td>
<td>9</td>
<td>7.5</td>
</tr>
<tr>
<td>set1-n30r1</td>
<td>24</td>
<td>24.4</td>
</tr>
<tr>
<td>set1-n30r2</td>
<td>116</td>
<td>69.7</td>
</tr>
<tr>
<td>set1-n30r3</td>
<td>46</td>
<td>40.1</td>
</tr>
<tr>
<td>set1-n30r4</td>
<td>31</td>
<td>19.3</td>
</tr>
<tr>
<td>set1-n30r5</td>
<td>78</td>
<td>64.3</td>
</tr>
<tr>
<td>set1-n30r6</td>
<td>30</td>
<td>29.6</td>
</tr>
<tr>
<td>set1-n30r7</td>
<td>29</td>
<td>18.0</td>
</tr>
<tr>
<td>set1-n30r8</td>
<td>1</td>
<td>10.2</td>
</tr>
<tr>
<td>set1-n30r9</td>
<td>2</td>
<td>11.3</td>
</tr>
<tr>
<td>set2-n30r0</td>
<td>9</td>
<td>8.5</td>
</tr>
<tr>
<td>set2-n30r1</td>
<td>24</td>
<td>23.8</td>
</tr>
<tr>
<td>set2-n30r2</td>
<td>51</td>
<td>31.7</td>
</tr>
<tr>
<td>set2-n30r3</td>
<td>46</td>
<td>39.4</td>
</tr>
<tr>
<td>set2-n30r4</td>
<td>33</td>
<td>22.1</td>
</tr>
<tr>
<td>set3-n30r0</td>
<td>8</td>
<td>3.1</td>
</tr>
<tr>
<td>set3-n30r1</td>
<td>56</td>
<td>9.4</td>
</tr>
<tr>
<td>set3-n30r2</td>
<td>4</td>
<td>0.7</td>
</tr>
<tr>
<td>set3-n30r3</td>
<td>242</td>
<td>80.3</td>
</tr>
<tr>
<td>set3-n30r4</td>
<td>820</td>
<td>568.6</td>
</tr>
</tbody>
</table>
Growth of Separating BDD for All but 3 Instances
Growth of Separating BDD for 2 Harder Instances
Growth of Separating BDD for Hardest Instance
Empirical Growth

• Separating BDD grows more or less linearly in all but one instance.
  – Somewhat superlinear in hardest instance.
  – Most BDDs never exceeded width of 100.
  – A width-1000 BDD can be processed in small fraction of a second.

• Hardest instance:
  – Width 16,496.
  – 820 iterations.
  – Final iteration processed in 2.9 seconds, including solution of subproblem.
Exploiting structure

- Include relaxation of subproblem in the master problem
  - Common technique in logic-based Benders.
- How? Add routing and scheduling variables to **bottom** of BDD in the master.
  - The Benders cuts will still contain only assignment variables.
  - By previous corollary, the rest of the BDD will be **unchanged** after separation.
Conclusions

• General separation algorithm is straightforward for BDDs.
  – Reduced separating BDD grows \textit{exponentially} in worst case.
    • Classical relaxation grows \textit{linearly}.
  – Empirically, BDD may grow \textit{linearly}
    • Almost always grows linearly in a representative application of logic-based Benders decomposition.
Conclusions

- So separation may have a useful role in BDD-based optimization.
  - But we must keep a close eye on growth rate.
  - Exploiting problem structure with variable ordering may reduce growth rate.