Decision Diagrams for Discrete Optimization

Willem-Jan van Hoeve

Tepper School of Business
Carnegie Mellon University

based on joint work with
David Bergman, Andre A. Cire, Sam Hoda, and John N. Hooker
Outline

• Motivation and background
  – multi-valued decision diagrams (MDDs)
• Constraint Programming with MDDs
• MDDs as Relaxations
• MDDs as Restrictions
• Conclusions
Decision Diagrams

- Binary Decision Diagrams were introduced to compactly represent Boolean functions [Lee, 1959], [Akers, 1978], [Bryant, 1986]
- Main operation: merge isomorphic subtrees of a given binary decision tree
- MDDs are multi-valued decision diagrams (i.e., for discrete variables)

\[ f(x_1, x_2, x_3) = -x_1 \cdot -x_2 \cdot -x_3 + x_1 \cdot x_1 \cdot x_2 + x_2 \cdot x_3 \]
Brief background

- Original application areas: circuit design, verification
- Usually Reduced Ordered BDDs/MDDs are applied
  - fixed variable ordering
  - minimal exact representation
- Recent interest from optimization community
  - cut generation [Becker et al., 2005]
  - 0/1 vertex and facet enumeration [Behle & Eisenbrand, 2007]
  - post-optimality analysis [Hadzic & Hooker, 2006, 2007]
- Interesting variant
  - approximate MDDs
    [H.R. Andersen, T. Hadzic, J.N. Hooker, & P. Tiedemann, CP 2007]
Exact MDDs for discrete optimization

(1) $x_1 + x_2 + x_3 \geq 1$
(2) $x_1 + x_4 + x_5 \geq 1$
(3) $x_2 + x_4 \geq 1$
Exact MDDs for discrete optimization

(1) \( x_1 + x_2 + x_3 \geq 1 \)
(2) \( x_1 + x_4 + x_5 \geq 1 \)
(3) \( x_2 + x_4 \geq 1 \)
Exact MDDs for discrete optimization

(1) \( x_1 + x_2 + x_3 \geq 1 \)
(2) \( x_1 + x_4 + x_5 \geq 1 \)
(3) \( x_2 + x_4 \geq 1 \)
(1) $x_1 + x_2 + x_3 \geq 1$
(2) $x_1 + x_4 + x_5 \geq 1$
(3) $x_2 + x_4 \geq 1$
Each path corresponds to a solution

(1) $x_1 + x_2 + x_3 \geq 1$
(2) $x_1 + x_4 + x_5 \geq 1$
(3) $x_2 + x_4 \geq 1$
Approximate MDDs

- Exact MDDs can be of exponential size in general
- Can we limit the size of the MDD and still have a meaningful representation?
  - Yes, first proposed by Andersen et al. [2007]: Limit the width of the MDD (the maximum number of nodes on any layer)

- This talk: applications to CP and IP
MDDs for Constraint Programming

Constraint Programming applies
• systematic search and
• inference techniques
to solve combinatorial problems

Inference mainly takes place through:
• **Filtering** provably inconsistent values from variable domains
• **Propagating** the updated domains to other constraints

\[
\begin{align*}
x_1 > x_2 \\
x_1 + x_2 &= x_3 \\
\text{alldifferent}(x_1, x_2, x_3, x_4)
\end{align*}
\]

\[x_1 \in \{2\}, \ x_2 \in \{1\}, \ x_3 \in \{3\}, \ x_4 \in \{0\}\]
Drawback of domain propagation

Observations:
- Communication between constraints only via variable domains
- Information can only be expressed as a domain change
- Other (structural) information that may be learned by a constraint is lost: it must be projected onto variable domains
- Potential solution space implicitly defined by Cartesian product of variable domains (very coarse relaxation)

This drawback can be addressed by communicating more expressive information, using MDDs

Explicit representation of more refined potential solution space

[Andersen et al. 2007]
Illustrative Example

AllEqual(x₁, x₂, x₃, x₄), all xᵢ binary

domain representation, size 2⁴

MDD representation, size 2
MDD-based constraint programming

• Maintain limited-width MDD
  – Serves as relaxation
  – Typically start with width 1 (initial variable domains)
  – Dynamically adjust MDD, based on constraints

• Constraint Propagation
  – Edge filtering: Remove provably inconsistent edges (those that do not participate in any solution)
  – Node refinement: Split nodes to separate edge information

• Search
  – As in classical CP, but may now be guided by MDD
Specific MDD propagation algorithms

- Linear equalities and inequalities  [Hadzic et al., 2008]
  [Hoda et al., 2010]
- *Alldifferent* constraints  [Andersen et al., 2007]
- *Element* constraints  [Hoda et al., 2010]
- *Among* constraints  [Hoda et al., 2010]
- Sequential scheduling constraints  [Hoda et al., 2010]
  [Cire & v.H., 2011]
- *Sequence* constraints (combination of *Amongs*)  [v.H., 2011]
- Generic re-application of existing domain filtering algorithm for any constraint type  [Hoda et al., 2010]
Case study: Among constraints

- Given a set of variables $X$, and a set of values $S$, a lower bound $l$ and upper bound $u$,

\[
\text{Among}(X, S, l, u) := l \leq \sum_{x \in X} (x \in S) \leq u
\]

“among the variables in $X$, at least $l$ and at most $u$ take a value from the set $S$”

- Applications in, e.g., sequencing and scheduling
- WLOG assume here that $X$ are binary and $S = \{1\}$
Example: MDD for Among

Exact MDD for Among($\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2$)
Goal: Given an MDD and an Among constraint, remove all inconsistent edges from the MDD (establish “MDD-consistency”)

Approach:
• Compute path lengths from the top node and from the bottom node
• Remove edges that are not on a path with lengths between lower and upper bound
• Complete (MDD-consistent) version
  – Maintain all path lengths; quadratic time
• Partial version (does not remove all inconsistent edges)
  – Maintain and check bounds (longest and shortest paths); linear time
Node refinement for Among

For each layer in MDD, we first apply edge filter, and then try to refine

- consider incoming edges for each node
- split the node if there exist incoming edges that are not equivalent (w.r.t. path length)

Example:

- We will propagate Among({x_1,x_2,x_3,x_4},{1},2,2) through a BDD of maximum width 3
Example

\[ \text{Among}\{\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2\} \]
Example

Among({x_1, x_2, x_3, x_4}, {1}, 2, 2)
Example

Among\(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2\)
Example

\[ \text{Among}\{x_1,x_2,x_3,x_4\},\{1\},2,2) \]
Experiments

- **Multiple among constraints**
  - 50 binary variables total
  - 5 variables per among constraint, indices chosen from normal distribution with uniform-random mean in $[1..50]$ and stdev 2.5, modulo 50
  - Classes: 5 to 200 among constraints (step 5), 100 instances per class

- **Nurse rostering instances** (horizon $n$ days)
  - Work 4-5 days per week
  - Max $A$ days every $B$ days
  - Min $C$ days every $D$ days
  - Three problem classes

- Compare width 1 (traditional domains) with increasing widths
Multiple Amongs: Backtracks

width 1 vs 4

width 1 vs 16
Multiple Amongs: Running Time

width 1 vs 4

width 1 vs 16
# Nurse rostering problems

<table>
<thead>
<tr>
<th>Size</th>
<th>Width 1</th>
<th></th>
<th>Width 4</th>
<th></th>
<th>Width 32</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BT</td>
<td>CPU</td>
<td>BT</td>
<td>CPU</td>
<td>BT</td>
<td>CPU</td>
</tr>
<tr>
<td>Class 1</td>
<td>40</td>
<td>61,225</td>
<td>55.63</td>
<td>8,138</td>
<td>12.64</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>175,175</td>
<td>442.29</td>
<td>5,025</td>
<td>44.63</td>
<td>11</td>
</tr>
<tr>
<td>Class 2</td>
<td>40</td>
<td>179,743</td>
<td>173.45</td>
<td>17,923</td>
<td>32.59</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>179,743</td>
<td>459.01</td>
<td>8,747</td>
<td>80.62</td>
<td>2</td>
</tr>
<tr>
<td>Class 3</td>
<td>40</td>
<td>91,141</td>
<td>84.43</td>
<td>5,148</td>
<td>9.11</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>882,640</td>
<td>2,391.01</td>
<td>33,379</td>
<td>235.17</td>
<td>55</td>
</tr>
</tbody>
</table>
Summary for MDD-based CP

- MDD provides substantial advantage over traditional domains for filtering multiple Among constraints
  - Strength of MDD can be controlled by the width
  - Wider MDDs yield greater speedups
  - Huge reduction in the amount of backtracking and solution time

- Intensive processing at search nodes can pay off when more structural information is communicated between constraints
Relaxation MDDs

Motivation and outline

- Limited width MDDs provide a (discrete) relaxation to the solution space
- Can we exploit MDDs to obtain bounds for discrete optimization problems?
Handling objective functions

Suppose we have an objective function of the form

$$\min \sum_i f_i(x_i)$$

for arbitrary functions $f_i$

In an exact MDD, the optimum can be found by a shortest $r$-$s$ path computation (edge weights are $f_i(x_i)$).

(1) $x_1 + x_2 + x_3 \geq 1$
(2) $x_1 + x_4 + x_5 \geq 1$
(3) $x_2 + x_4 \geq 1$
Approach

• Construct the relaxation MDD using a top-down compilation method
• Find shortest path → provides bound B
• Extension to an exact method
  1. Isolate all paths of length B, and verify if any of these paths is feasible*
  2. if not feasible, set B := B + 1 and go to 1
  3. otherwise, we found the optimal solution

* Feasibility can be checked using MDD-based CP
Case Study: Set covering problem

• Given set $S=\{1,\ldots,n\}$ and subsets $C_1,\ldots,C_m$ of $S$
• Find a subset $X$ of $S$ with minimum cardinality such that $|C_i \cap X| \geq 1$ for all $i=1,\ldots,m$

$$\min \sum_j x_j$$

s.t.  $$\sum_{j \in C_i} x_j \geq 1 \quad \text{for all } i=1,\ldots,m$$

$x_1,\ldots,x_n$ binary
Exact top-down compilation

\[(1) \ x_1 + x_3 \geq 1 \]
\[(2) \ x_2 + x_4 \geq 1 \]

{Indices of the constraints that still need a 1}
Equivalence test for set covering

(1) \( x_1 + x_3 \geq 1 \)
(2) \( x_2 + x_4 \geq 1 \)

Relaxation MDD: merge non-equivalent nodes when the given width is exceeded
\(1\) \(x_1 + x_2 + x_3 \geq 1\)
\(2\) \(x_1 + x_4 + x_5 \geq 1\)
\(3\) \(x_2 + x_4 + x_6 \geq 1\)

\{Indices of the constraints that still need a 1\}

Exact MDD

Relaxation MDD (width ≤ 3)
Exact MDD

(1) \(x_1 + x_2 + x_3 \geq 1\)
(2) \(x_1 + x_4 + x_5 \geq 1\)
(3) \(x_2 + x_4 + x_6 \geq 1\)

Relaxation MDD (width \(\leq 3\))

\(\emptyset\) \{3\} \{3\} \{2\} \{1,2,3\}

\{Indices of the constraints that still need a 1\}
Exact MDD

Relaxation MDD (width $\leq 3$)

(1) $x_1 + x_2 + x_3 \geq 1$
(2) $x_1 + x_4 + x_5 \geq 1$
(3) $x_2 + x_4 + x_6 \geq 1$

Indices of the constraints that still need a 1
Exact MDD

Relaxation MDD (width ≤ 3)

(1) $x_1 + x_2 + x_3 \geq 1$
(2) $x_1 + x_4 + x_5 \geq 1$
(3) $x_2 + x_4 + x_6 \geq 1$

{Indices of the constraints that still need a 1}
Exact MDD

Relaxation MDD (width ≤ 3)

(1) \( x_1 + x_2 + x_3 \geq 1 \)
(2) \( x_1 + x_4 + x_5 \geq 1 \)
(3) \( x_2 + x_4 + x_6 \geq 1 \)

\[ \begin{align*}
x_1 & \\
\vdots & \\
x_6 & 
\end{align*} \]

\{Indices of the constraints that still need a 1\}
Exact MDD

\[ (0,0,1,0,1,1) \]

Relaxation MDD (width \( \leq 3 \))

\[
\begin{align*}
(1) & \quad x_1 + x_2 + x_3 \geq 1 \\
(2) & \quad x_1 + x_4 + x_5 \geq 1 \\
(3) & \quad x_2 + x_4 + x_6 \geq 1
\end{align*}
\]
Exact MDD

\[ r \]

Relaxation MDD (width \( \leq 3 \))

\[ r \]

(1) \( x_1 + x_2 + x_3 \geq 1 \)
(2) \( x_1 + x_4 + x_5 \geq 1 \)
(3) \( x_2 + x_4 + x_6 \geq 1 \)

\[ x_1 \]

\[ x_2 \]

\[ x_3 \]

\[ x_4 \]

\[ x_5 \]

\[ x_6 \]

\[ (0,0,0,1,1,1,1) \]
min $f(x) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6$
Exact MDD

(1) \( x_1 + x_2 + x_3 \geq 1 \)
(2) \( x_1 + x_4 + x_5 \geq 1 \)
(3) \( x_2 + x_4 + x_6 \geq 1 \)

\[ r \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad s \]

\( f(x^*) = 2 \)

\( \min f(x) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \)

Relaxation MDD (width \( \leq 3 \))

\( f(x^*) = 1 \)
Tightening the Lower Bound

• Value extraction method
  – Given: an MDD relaxation, M
  – Given: a valid lower bound, v
  – Extract all paths in M that correspond to solutions with objective function value equal to v in the form of another MDD $M|_{z=v}$

• Creating $M|_{z=v}$ can be done efficiently

• Apply MDD-based CP to $M|_{z=v}$ in order to either
  – Increase v to v+1 (if no solution exists)
  – Find a feasible (and optimal) solution
Experimental Results

• Investigate whether relaxation MDDs are able to capture and exploit problem structure
  – We consider structured set covering problems

• Purest structure: all constraints are defined on consecutive variables
  – TU constraint matrix; easy for IP
  – exact MDD has bounded width; also easy for MDD
Instance Generation

• We generated random instances
  – Fix number of variables per constraint, $k$
  – Vary the bandwidth, $b_w$
  – Randomly assign a 0 to $b_w$ – $k$ ones in the bandwidth

• Destroys both the TUM property for IP and the bounded width property for MDD
Computational Results

- 250 variables, 20 instances, $k = 20$, $b_w \in \{22, ..., 44\}$
- Compare 3 different solution methods
  - Pure-IP (CPLEX)
  - Pure-MDD (Value Extraction)
  - Hybrid (1/10 solution time given to pure-MDD and then pass bound to CPLEX)
Number of Instances Solved (1 min.)

![Graph showing number of instances solved vs. bandwidth with lines for IP, MDD, and HYBRID methods.](image-url)
Average Ending Lower Bound (1 min)
Larger Instances

- 500 variables, 5 instances, \( k = 20 \), \( b_w \in \{22, \ldots, 25\} \)
Restriction MDDs
• Restriction MDDs represent a subset of feasible solutions
  – we require that every r-s path corresponds to a feasible solution
  – but not all solutions need to be represented
• Goal: Use restriction MDDs as a heuristic to find good feasible solutions
Creating Restriction MDDs

Using an exact top-down compilation method, we can create a limited-width restriction MDD by

1. merging nodes, or
2. deleting nodes

while ensuring that no solution is lost
Node merging by example

Restriction MDD (width ≤ 3)

(1) $x_1 + x_2 + x_3 \geq 1$
(2) $x_1 + x_4 + x_5 \geq 1$
(3) $x_2 + x_4 + x_6 \geq 1$

{Indices of the constraints that still need a 1}
Node merging by example

Restriction MDD (width $\leq 3$)

(1) $x_1 + x_2 + x_3 \geq 1$
(2) $x_1 + x_4 + x_5 \geq 1$
(3) $x_2 + x_4 + x_6 \geq 1$

{Indices of the constraints that still need a 1}

∅  {3}{1,2,3}  {2}  {1,2,3}
Node merging heuristics

- **Random**
  - select two nodes \( \{u_1, u_2\} \) uniformly at random

- **Objective-driven**
  - select two nodes \( \{u_1, u_2\} \) such that
    \[
    f(u_1), f(u_2) \geq f(v) \text{ for all nodes } v \neq u_1, u_2 \text{ in the layer}
    \]

- **Similarity**
  - select two nodes \( \{u_1, u_2\} \) that are ‘closest’
  - problem dependent (or based on semantics)
  - for our set covering example: symmetric difference
Node deletion by example

Restriction MDD (width ≤ 3)

(1) \(x_1 + x_2 + x_3 \geq 1\)
(2) \(x_1 + x_4 + x_5 \geq 1\)
(3) \(x_2 + x_4 + x_6 \geq 1\)

\[
\begin{array}{cccc}
\emptyset & \{3\} & \{2\} & \{1,2,3\} \\
\{3\} & r & \{1,2,3\} \\
\end{array}
\]

{Indices of the constraints that still need a 1}
Node deletion heuristics

• Random
  – select node $u$ uniformly at random

• Objective-driven
  – select node $u$ such that
    \[ f(u) \geq f(v) \text{ for all nodes } v \neq u \text{ in the layer} \]

• Information-driven
  – for set covering: select node $u$ such that
    \[ I(u) \geq I(v) \text{ for all nodes } v \neq u \text{ in the layer} \]
    where $I(u)$ is the set of constraints that still need a 1
Preliminary Experimental Results

• Goals
  – obtain insight in the relative strength of the different restriction heuristics
  – compare to well-known greedy heuristic [Chvátal, 1979]

• Randomly generated set covering instances
  – n variables and m constraints
  – n, m ∈ {25, 50}, with 25 instances per setting
  – unit cost instances and random cost instances (costs are uniform-randomly drawn from {1,...,20})

• MDD widths: 10, 25, 50, 100
# Unit costs

<table>
<thead>
<tr>
<th>n</th>
<th>m width</th>
<th>Merge</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>rnd</td>
<td>obj</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>1.47</td>
<td>1.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.32</td>
<td>1.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.23</td>
<td><strong>1.00</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.14</td>
<td><strong>1.00</strong></td>
</tr>
<tr>
<td>25</td>
<td>50</td>
<td>1.41</td>
<td>1.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.33</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.26</td>
<td>1.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.23</td>
<td>1.02</td>
</tr>
<tr>
<td>50</td>
<td>25</td>
<td>1.50</td>
<td>1.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.38</td>
<td>1.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.29</td>
<td>1.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.21</td>
<td>1.01</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>1.67</td>
<td>1.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.60</td>
<td>1.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.55</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.48</td>
<td><strong>1.04</strong></td>
</tr>
</tbody>
</table>

Averages over 25 instances

Compilation time (obj-based) is around 0.05s
## Random costs

<table>
<thead>
<tr>
<th>n</th>
<th>m</th>
<th>width</th>
<th>Merge</th>
<th></th>
<th>Delete</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>rnd</td>
<td>obj</td>
<td>simil</td>
<td>rnd</td>
<td>obj</td>
<td>info</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>10</td>
<td>1.66</td>
<td>1.12</td>
<td>1.36</td>
<td>1.65</td>
<td>1.10</td>
<td>1.53</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td>1.48</td>
<td><strong>1.04</strong></td>
<td>1.20</td>
<td>1.37</td>
<td><strong>1.04</strong></td>
<td>1.37</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td>1.31</td>
<td><strong>1.01</strong></td>
<td>1.14</td>
<td>1.23</td>
<td><strong>1.01</strong></td>
<td>1.26</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td>1.17</td>
<td><strong>1.00</strong></td>
<td>1.08</td>
<td>1.11</td>
<td><strong>1.00</strong></td>
<td>1.16</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>50</td>
<td>10</td>
<td>1.79</td>
<td>1.14</td>
<td>1.41</td>
<td>1.80</td>
<td><strong>1.13</strong></td>
<td>1.64</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td>1.60</td>
<td><strong>1.07</strong></td>
<td>1.33</td>
<td>1.51</td>
<td><strong>1.07</strong></td>
<td>1.52</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td>1.57</td>
<td><strong>1.04</strong></td>
<td>1.29</td>
<td>1.41</td>
<td><strong>1.04</strong></td>
<td>1.42</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td>1.42</td>
<td><strong>1.02</strong></td>
<td>1.17</td>
<td>1.36</td>
<td><strong>1.02</strong></td>
<td>1.34</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>25</td>
<td>10</td>
<td>1.76</td>
<td><strong>1.09</strong></td>
<td>1.46</td>
<td>1.83</td>
<td><strong>1.09</strong></td>
<td>1.51</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td>1.57</td>
<td><strong>1.02</strong></td>
<td>1.35</td>
<td>1.63</td>
<td><strong>1.02</strong></td>
<td>1.38</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td>1.46</td>
<td><strong>1.01</strong></td>
<td>1.28</td>
<td>1.46</td>
<td><strong>1.01</strong></td>
<td>1.26</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td>1.30</td>
<td><strong>1.00</strong></td>
<td>1.20</td>
<td>1.27</td>
<td><strong>1.00</strong></td>
<td>1.15</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>10</td>
<td>2.00</td>
<td>1.25</td>
<td>1.55</td>
<td>1.94</td>
<td><strong>1.21</strong></td>
<td>1.87</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td>1.87</td>
<td><strong>1.13</strong></td>
<td>1.42</td>
<td>1.74</td>
<td><strong>1.13</strong></td>
<td>1.72</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td>1.83</td>
<td><strong>1.09</strong></td>
<td>1.38</td>
<td>1.68</td>
<td><strong>1.09</strong></td>
<td>1.63</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td>1.74</td>
<td><strong>1.05</strong></td>
<td>1.30</td>
<td>1.60</td>
<td><strong>1.05</strong></td>
<td>1.52</td>
<td></td>
</tr>
</tbody>
</table>

Averages over 25 instances.

Compilation time (obj-based) is around 0.05s.
# MDDs versus greedy heuristic

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>MDD+</th>
<th>MDD-</th>
<th>MDD+</th>
<th>MDD-</th>
<th>MDD+</th>
<th>MDD-</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>25</td>
<td>5</td>
<td>20</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td></td>
<td>21</td>
<td>4</td>
<td>0</td>
<td>13</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td>21</td>
<td>4</td>
<td>0</td>
<td>15</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>50</td>
<td>5</td>
<td>23</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td></td>
<td>24</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td>24</td>
<td>1</td>
<td>0</td>
<td>19</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>25</td>
<td>5</td>
<td>15</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>25</td>
<td>18</td>
<td>6</td>
<td>1</td>
<td>20</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td>21</td>
<td>4</td>
<td>0</td>
<td>23</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>5</td>
<td>16</td>
<td>8</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>21</td>
</tr>
<tr>
<td>25</td>
<td>21</td>
<td>4</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td>24</td>
<td>1</td>
<td>0</td>
<td>18</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
Conclusions

• Limited-width MDDs can be a very useful tool for discrete optimization
  – The maximum width provides a natural trade-off between computational efficiency and strength
  – Powerful inference mechanism for constraint propagation
  – Generic discrete relaxation and restriction method for MIP-style problems

• Many open questions
  – MDD variable ordering, interaction with search, formal characterizations, ...