MINCE: A Static Global Variable-Ordering for SAT and BDDs

Fadi A. Aloul, Igor L. Markov, Karem A. Sakallah

University of Michigan

Outline

- Hypergraph Terminology
- Motivating Example
- Multilevel Partitioning
- MINCE Algorithm
- Experimental Results
- Conclusions

© 2001 Fadi A. Aloul, University of Michigan
**Linearly-Ordered Hypergraphs**

- Given a hypergraph with \( V \) vertices and \( E \) hyperedges with a linear vertex order...
  - **Span** of hyperedge: difference between the greatest and smallest vertices connected by the same hyperedge
  - **i-th cut**: number of edges crossing vertex \( i + 0.5 \)
  - **Cutwidth**: maximum cut of all vertices \( i \), \( i \in (0,\ldots,n-1) \)
  - An objective of vertex ordering: identify a linear vertex order that minimizes the span and cutwidth of the instance

```
\[ f(a,b,c,d,e) = (a + d + e) \land (b + d) \land (c + e) \]
```

**Bad vs. Good Vertex Orderings**

- Total Span = 8  Cutwidth = 3  Total Span = 4  Cutwidth = 1

How does vertex reordering help?

Converting CNF Formulas to Hypergraphs:

- Variables \( \Rightarrow \) Vertices
- Clauses \( \Rightarrow \) Hyperedges

© 2001 Fadi A. Abou, University of Michigan
Related Work

- Circuits with small cutwidth are theoretically “easy” for SAT [Prasad et al. 99]
- Sizes of BDDs are correlated with circuit cutwidth [Berman 91, McMillan 92]
- Extracted BDD variable orderings from linear spectral hypergraph placement [Wood et al. 98]
- This work considers average cutwidth instead of maximum cutwidth

Example

Hole-7 Instance
(clauses in red)

Original Variable Order  MINCE Variable Order
Observation: Crossing Minimization

\[ \text{TotalSpan} = \sum_{e \in E} \text{span}(e) = \sum_{e \in E} \sum_{s \in E} 1 = \text{#xings} = \sum_{s \in E} \sum_{i \in I} 1 = \sum_{i \in I} \text{cut}(i) \]

\[ \text{AverageSpan} = \frac{\sum_{e \in E} \text{span}(e)}{E} \]

\[ \text{AverageCut} = \frac{\sum_{i \in I} \text{cut}(i)}{V - 1} = E \cdot \frac{\sum_{e \in E} \text{span}(e)}{E} = \frac{E}{V} \cdot \frac{V}{V - 1} \cdot \text{AverageSpan} \]

\[ \text{AverageCut} = \frac{C}{V} \cdot \text{AverageSpan} \quad \text{Min. AverageCut} \leftrightarrow \text{Min. AverageSpan} \]

Known from VLSI placement:
Recursive Min-cut Bisection \(\Rightarrow\) Min. Total Net Length in LinPlacement

Linear Placement

- Net length objective (aka “bounding box”)
  - For CNF instances, translates into \(\Sigma\) clause span
- 30+ years of placement research
  - Recursive bisection a leading method
  - Applied to SAT in this work
- CAPO: Efficient hypergraph placement software
  - Caldwell, Kahng and Markov [DAC 00]
  - Based on Recursive Min-cut Bisection
  - Multilevel Fiduccia-Mattheyses (FM)
  - Open-source, free:
    - [http://vlsicad.cs.ucla.edu/software/PDtools](http://vlsicad.cs.ucla.edu/software/PDtools)
  - Runs in: \(\Theta(N \log^2 N)\) \(\quad\), \(N\) is size of input

© 2001 Fadi A. Aklou, University of Michigan
Min-Cut MLFM Partitioning

- **MLPart**: Efficient min-cut hypergraph partitioner
  - Caldwell, Kahng and Markov [ASPDAC 00]
  - Outperforms hMetis (Karypis et al. [DAC 97])
  - Runs in: \( \Theta(N \log N) \)
  - Called by CAPO

**Basic Idea:**
- Group original variables
- Induce clustered hypergraphs
- Partition clustered hypergraphs
- Refine partitioned hypergraphs
- Partition & refinement by Fiduccia-Mattheyses

---

**MINCE - Flow Diagram**

```
MINCEFlow
  ── CNF instance
  │   Hypergraph
  │   Linear Min-cut Placement by Recursive MLFM Partitioning, e.g. CAPO
  │   Variable ordering for CNF
  │   Preprocessed CNF instance
  ── SAT Solver
  ── BDD Engine
```

© 2001 Fadi A. Aloul, University of Michigan
Experimental Setup

- SAT engine: GRASP SAT Solver
- BDD engine: CUDD Package
- Time-out limit: 10,000 seconds
- Memory limit: 500 Mb
- Platform: 333 MHz Pentium II with Linux
- Benchmarks: DIMACS, N-Queens, ISCAS89

SAT Results

DIMACS Benchmarks®

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Total Runtime (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLCS</td>
<td>216</td>
</tr>
<tr>
<td>MSOS</td>
<td>218</td>
</tr>
<tr>
<td>Fixed</td>
<td>218</td>
</tr>
<tr>
<td>MSTS</td>
<td>219</td>
</tr>
<tr>
<td>DLIS</td>
<td>219</td>
</tr>
<tr>
<td>MNCE</td>
<td>222</td>
</tr>
</tbody>
</table>
BDD Results

ISCAS 89 Benchmarks

Best- vs. Worst-case Performance

- SAT/BDD
  - Worst-case: \( \exp \)  Best-case: \( \Theta(N) \)
- Recursive min-cut bisection placement
  - Worst-case: \( \Theta(N \log^2 N) \)  Best-case: \( \Theta(N \log^2 N) \)
- Very easy problem instances
  - DLL/BDD run in near-linear time
  - Vertex ordering only slows DLL/BDD
  - MINCE is not helpful for easy instances

© 2001 Fadi A. Aloul, University of Michigan
Conclusions

- MINCE is useful in capturing the structural properties of CNF instances
- MINCE ordering is very effective in reducing SAT runtime time and BDD runtime/memory requirements
- The ordering is easily generated in a preprocessing step
- No source code modification needed
- Tools are publicly available!

Future Work

- Further improving the MINCE algorithm
- Accounting for polarities of literals in hypergraphs
- Applying the ordering to symbolic simulation
- Tracking empirical correlation between problem complexity and its cutwidth

- Check out MINCE @: http://andante.eecs.umich.edu/mince