Faster SAT and Smaller BDDs via Common Function Structure

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

University of Michigan

Motivation

Hole-7 Instance
(clauses in red)

Original Variable Order  “New” Variable Order
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 $i, i \in (0, ..., n-1)$
  - An objective of vertex ordering: identify a linear vertex order that minimizes the span and cutwidth of the instance
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

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

Related Work

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

- This work considers \textit{average cutwidth} instead of maximum \textit{cutwidth}
Example

Hole-7 Instance (clauses in red)

Original Variable Order

MINCE Variable Order

Observation: Crossing Minimization

\[
TotalSpan = \sum_{e \in E} \text{span}(e) = \sum_{c \in C} \sum_{x \in c} 1 = \# \text{xings} = \sum_{x \in C} \sum_{c \in x} 1 = \sum_{i=0}^{V-1} \text{cut}(i)
\]

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

\[
AverageCut = \frac{\sum_{i=0}^{V-1} \text{cut}(i)}{V-1} = \frac{E \cdot \sum_{e \in E} \text{span}(e)}{V \cdot E} = \frac{E \cdot \sum_{x \in C} \sum_{c \in x} 1}{V \cdot \sum_{c \in C} \sum_{x \in c} 1} = \frac{1}{V} \cdot \text{AverageSpan}
\]

\[
AverageCut = \frac{C}{V} \cdot \text{AverageSpan}
\]

Min. AverageCut $\leftrightarrow$ 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 sum of 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)
  - Runs in: $\Theta(N \log^2 N)$, $N$ is size of input

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

*By G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar*
MINCE - Flow Diagram

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
DIMACS Benchmarks*

*Except f, g, par32

Selected DIMACS Instances
SAT Results

Selected NQueens Instances

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

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

- Dramatic speedup improvements possible
- 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