Efficient Computation of Practical Variable Orderings

Kalev Kask
2010-11-18
Basic Ideas

- Use greedy incremental MinFill heuristic
- Parallel computation
- Pre-compute/pre-allocate as much as possible
- Incremental updates during search
- Abandon single computation as early as possible if it cannot improve upon previous
Algorithm Outline

• Input:
  – Problem
  – m = #threads
  – Timelimit
  – Pool size for random picking

• Output: min-width and min-complexity orderings

• Algorithm
  – Preprocess the problem, producing problem*
    • Often, the size of problem* is half or less than that of the original problem
  – On m threads, until timelimit is reached, execute a single run of the ordering algorithm
    • Start with a copy of problem* as the starting point
Preprocessing

- Eliminate singleton domain variables from functions
- Run SAT-based singleton-consistency, pruning domains of variables
- Build a primal graph of the problem
  - sorted adj-var list for each variable
- Compute degree, MinFill scores for all variables
- Split variables into 3 disjoint sets
  - Degree(X) <= 1
  - MinFillScore = 0
  - The rest
- Execute single run of the ordering algorithm, until there are no Degree<=1 and MinFillScore=0 variables left
- Resulting problem* is the starting point of all subsequent calls to the ordering algorithm

Hypothesis: MFS=0 var elimination order does not matter; the result is always the same.
Variable lists

• Each variable belongs to exactly 1 list
  – Order
  – Degree(X) <= 1
  – MinFillScore = 0
  – Everything else

• Should be able to add/remove a variable to/from a list is const time
Basic Order Computation

• Iterative algorithm, eliminating single variable at each iteration
• During each iteration, check if width/complexity is over the limit; if yes, exit. This can be turned off.
• Take variables from lists in this order
  – Degree <= 1
  – MinFillScore = 0
  – Everything else
    • Draw variable randomly from a pool of given size of best (MinFillScore lowest) variables
• Add variable to the order
• Update the graph
Updating MinFill scores

• Variables whose degree may change
  – Variable u adjacent to X (add -1 to u)
  – Variables u and v adjacent to X but not to each other (add +1 to both)

• Variables whose MinFill score may change
  – edge u→X is deleted : variable u is adjacent to X and v, and v is not adjacent to X (add -1 to u)
    • Note v is outside the circle
  – edge u→v is added : Variable w adjacent to u but not v (add +1 to u)
    • Note w is outside the circle
  – edge u↔v is added : Variable w adjacent to both u/v (add -1 to w)
    • Note w is on or outside the circle
Updating the graph

1. Update neighbors of X (scan adj-var list of X).
   A. For each neighbor u of X, do
      • Remove edge u→X
      • Add edge u→v for each variable v adjacent to X but not u
         — Remember edges added
      • Update degree of u for each add/remove
      • Update \textit{MinFill} score of u due to edge removal

2. Update \textit{MinFill} scores due to edge addition

3. Update lists of variables (neighbors u of X and variables w outside the circle)
   – General → \textit{MinFillScore}=0 → Degree \leq 1

4. Remove all edges adjacent to X

• Complexity of 1A is (2*deg) since adj-var lists are sorted.
• Complexity of 1 is 2*deg^2.
• Naïve implementation of 1 would be deg^3
   – Enumerate all pairs of neighbors (deg^2)
   – For each pair, check if edge between exists, add if needed (deg)
• Complexity of 2 is fill*(2*deg).
Pool picking

type4_100_23, time limit=0:30:0 (1800), pick : score^γ

<table>
<thead>
<tr>
<th></th>
<th>0.1</th>
<th>0.5</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>49 (1 3 17 23 83 174 178 802 934 1796)</td>
<td>49 (1 2 2 2 15 18 19 46 89)</td>
<td>49(1 2 2 2 5 6 10 30 73 527 834 1094)</td>
</tr>
<tr>
<td>8</td>
<td>50(1 2 2 2 2 6 10 53 160 183 363 366 395 408 458 952)</td>
<td>50(1 2 2 2 3 5 6 8 14 21 28 50 61 234 566)</td>
<td>51(1 2 2 3 3 4 5 6 9 10 34 156 976)</td>
</tr>
<tr>
<td>12</td>
<td>51(1 3 3 5 13 15 288 510 542)</td>
<td>51(1 3 3 4 86 307 750 1129)</td>
<td>52(1 3 4 11 18 63 193 1013 1194)</td>
</tr>
</tbody>
</table>
**Pool picking**

type4_110_25, time limit=0:30:0 (1800), pick : score^γ

<table>
<thead>
<tr>
<th></th>
<th>0.1</th>
<th>0.5</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>49 (2 2 4 16 20 21 47 58 129 201)</td>
<td>49 (1 2 3 4 6 7 14 48 196 729)</td>
<td>47(1 2 2 2 2 3 4 5 7 36 86 101 393)</td>
</tr>
<tr>
<td>8</td>
<td>49(1 3 3 4 5 5 6 7 10 40 40 52 248 354 409 1249)</td>
<td>48(1 3 3 4 14 17 21)</td>
<td>48(1 3 3 3 3 4 6 7 16 19 26 85 138 184 422 1795)</td>
</tr>
<tr>
<td>12</td>
<td>48(1 4 4 7 7 41 67 68 77 440 692 1070 1711)</td>
<td>49(1 3 3 4 5 6 10 22 54 80 322 458 1536)</td>
<td>49(1 3 3 4 5 6 21 22 48 93 162 249 653 1528)</td>
</tr>
</tbody>
</table>
## Pool picking

**type4_140_23, time limit=0:30:0 (1800), pick : score^γ**

<table>
<thead>
<tr>
<th></th>
<th>0.1</th>
<th>0.5</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4</strong></td>
<td>58 (2 4 4 5 5 5 8 17 18 27 185)</td>
<td>59 (2 4 4 4 5 102 132)</td>
<td>60(2 5 7 21 33 34 71 619 641 660 1168)</td>
</tr>
<tr>
<td><strong>8</strong></td>
<td>59(2 5 5 6 8 10 54 77 108 131 390 698 920 1623)</td>
<td>59(2 6 6 6 6 6 7 8 30 81 129 211 271 561 1371)</td>
<td>59(2 6 6 6 7 8 9 16 88 495 1071)</td>
</tr>
<tr>
<td><strong>12</strong></td>
<td>59(2 14 26 250 884)</td>
<td>60(1 6 6 7 7 9 12 13 16 30 139 405 621 1238)</td>
<td>60(1 6 6 8 8 17 91 267 423 993)</td>
</tr>
</tbody>
</table>