Bucket Elimination With External Memory

Kalev Kask
Orbit Technology Corporation
ICS: automated reasoning group

joint work with Rina Dechter
Overview

• Motivated by the work of Richard Korf on using external memory for heuristic search
• Bucket Elimination is a general algorithm for automated reasoning.
• Fast and efficient, when applicable.
• In practice, it is memory constrained.
Bucket Elimination

Variable order: C, A, B, E, F

Bucket F: \( P(F|C,E) \)
Bucket E: \( P(B|E), P(E), h(C,E) \)
Bucket B: \( P(C|A,B), h(B,C) \)
Bucket A: \( P(A), h(A,C) \)
Bucket C: \( h(A) \)
Bucket Elimination

- Bucket Elimination = Tree Decomposition algorithm on a join-tree where $|\text{separator}(u,v)| = |u|-1$. 
Bucket Elimination Basic Step

• Bucket u
  – functions $f_1, \ldots, f_m$
  – variables $X_1, \ldots, X_n$
  – domain size of each variable is $k$

• Compute function $h(sep(u,v))$: combine all functions $f_i$, eliminate variable $u$-$sep(u,v)$
  – Most basic op: computation of a single entry in the table

• Table of $h$ has $k^{|sep(u,v)|}$ entries
Bucket Elimination

• Each function is represented as a table
• Space/time exponential in induced width
  \[ O(k^{1+w^*}) \]
• Typically runs in a few seconds; it will either solve the problem or run out of memory
• In practice limited by memory
  – Each table used (written/read) once
  – Unsuitable for problems with large domain sizes, except for very small \( w^* \)
BE vs BEEM

• For binary variables (k=2), when w*=20, table has $2^{20}=1,048,576$ entries; double-precision floating point is 8 bytes -> table is 8 MB
• When k=3, table is 28 GB
• Assume : BE has 1 GB, BEEM has 1TB
  – If k=2, BEEM can handle w* larger by 10 compared to BE
    • $\log_k(1000)=10$
  – If k=3, BEEM can handle w* larger by 6.3
RAM vs Disk Memory

- **RAM**
  - Bandwidth 3.2/6.4 GB/sec

- **Disk**
  - Seek time: 9ms
  - Sequential transfer rate: 100 MB/sec
  - Non-sequential transfer rate: bad
BEEM Basic Design Elements

• Divide each function into blocks
  – Compute/Save/Load a block at a time
• Minimize block swapping
  – Optimized function table indexing
• Multi-threading
Block Computation

• An entire table will not fit in memory
• Saving/loading each entry will cause disk I/O
• Each disk I/O will read/write a minimum-size block (e.g. 512 bytes)
• Solution:
  – Divide function table into blocks
  – Save each block as a separate file
  – When computing, compute entire block and then save
  – When using table, load entire block
Computing FTB size

Memory per thread = (RAM - original function space) / # threads
Computing FTB size

• Assume 1 block per neighbor is kept in memory
• Sort nodes in the bucket tree in decreasing order of degree of the node
• Process nodes in this order:
  – If function table size is trivial (e.g. 8K),
    • block size = function size -> 1 block
  – Otherwise,
    • Let k neighbors already have FTB size set
    • FTB size = (memory per thread - $\sum_{j=1\ldots k} |FTB_j|$)/(degree-k)
    • Minimum # of blocks = # threads
(Function Table Indexing

Typical table indexing: \( f(X_1, X_2, X_3) = \sum_Y (h_1(Y, X_3, X_1) h_2(X_3, Y)) \), \( k=3 \)

\[
\begin{array}{c|c}
\hline
h_1(Y, X_3, X_1) & f(X_1, X_2, X_3) \\
\hline
... & ... \\
3 & 010 \\
4 & 011 \\
5 & 012 \\
6 & 020 \\
7 & 021 \\
8 & 022 \\
9 & 100 \\
10 & 101 \\
11 & 102 \\
12 & 110 \\
13 & 111 \\
14 & 112 \\
15 & 120 \\
16 & 121 \\
17 & 122 \\
18 & 200 \\
19 & 201 \\
20 & 202 \\
21 & 210 \\
22 & 211 \\
23 & 212 \\
24 & 220 \\
... & ... \\
\hline
\end{array}
\]

\[
\begin{array}{c|c}
\hline
h_2(X_3, Y) &  \\
\hline
0 & 00 \\
1 & 01 \\
2 & 02 \\
3 & 10 \\
4 & 11 \\
5 & 12 \\
6 & 20 \\
7 & 21 \\
8 & 22 \\
\hline
\end{array}
\]
Reordering Function Scopes

- Given $f$, reorder $h_1$ and $h_2$, such that
  - Variable being eliminated is the last variable
  - Order of $h_1/h_2$ agrees with $f$

<table>
<thead>
<tr>
<th>$h_1(X_1,X_3,Y)$</th>
<th>$f(X_1,X_2,X_3)$</th>
<th>$h_2(X_3,Y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>010</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>011</td>
<td>00</td>
</tr>
<tr>
<td>5</td>
<td>012</td>
<td>000</td>
</tr>
<tr>
<td>6</td>
<td>020</td>
<td>0001</td>
</tr>
<tr>
<td>7</td>
<td>021</td>
<td>0002</td>
</tr>
<tr>
<td>8</td>
<td>022</td>
<td>0002</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>0002</td>
</tr>
<tr>
<td>10</td>
<td>101</td>
<td>0002</td>
</tr>
<tr>
<td>11</td>
<td>102</td>
<td>0002</td>
</tr>
<tr>
<td>12</td>
<td>110</td>
<td>0002</td>
</tr>
<tr>
<td>13</td>
<td>111</td>
<td>0002</td>
</tr>
<tr>
<td>14</td>
<td>112</td>
<td>0002</td>
</tr>
<tr>
<td>15</td>
<td>120</td>
<td>0002</td>
</tr>
<tr>
<td>16</td>
<td>121</td>
<td>0002</td>
</tr>
<tr>
<td>17</td>
<td>122</td>
<td>0002</td>
</tr>
<tr>
<td>18</td>
<td>200</td>
<td>0002</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>011</td>
<td>0002</td>
</tr>
<tr>
<td>5</td>
<td>012</td>
<td>0002</td>
</tr>
<tr>
<td>6</td>
<td>020</td>
<td>0002</td>
</tr>
<tr>
<td>7</td>
<td>021</td>
<td>0002</td>
</tr>
<tr>
<td>8</td>
<td>022</td>
<td>0002</td>
</tr>
<tr>
<td>9</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Reordering Function Scopes

• Reorder scopes of all functions in the bucket tree
  – Start from the top
  – Order each child node wrt its parent node
  – Applies to new functions only
Multi-Threading

• Single-thread problem
  – Disk I/O slow; CPU utilization will be low

• Solution
  – since CPUs have many cores (execution units), and
  – since each entry in the function table can be computed independently, therefore each block can be computed independently.
  – use many threads for computation
    • one thread per block
    • as one thread waits for disk I/O, other threads can compute
Worker thread

• Enumerate all entries in the FTB
  – Based on the index of the entry in output table, compute indexes in the input tables needed for this
  – Check, for each input table, if the block corresponding to the index is in memory
  – If not, tell the scheduler to unload the current block and load the new block
  – Compute the output entry

• Save FTB
Scheduler

- Create computation order, bottom up on the bucket tree
- Keep track of which FTBs are
  - Computed
  - In memory
- Assign tasks (FTBs to compute) to worker threads
Experimental Evaluation

• Compare BEEM against VEC
  – Both general purpose exact algorithms
• VEC was shown very competitive at UAI-2008 solver competition
• Benchmarks : Bayesian networks derived from genetics (pedigree/linkage networks)
• Compute $P(e)$
VEC
(Variable Elimination and Conditioning)

• Exact algorithm

• Idea :
  – Variable Elimination + Conditioning (Pearl ’88)
    • Condition on variables until the tree-width of the remaining problem is “reasonable” (variable elimination needs no more than 1GB of space on the remaining problem)
  – SAT based singleton consistency
VEC Algorithm

• Algorithm (Network P)
  – Reduction Step (Input: P, Output: P’)
    • Convert the zero probabilities in P to a SAT problem F
    • For each variable-value pair X=a
      – if (F and X=a) has no solutions (use minisat Een and Sörensson 06)
        » Remove X=a from P
    – If the reduced network P’ has a “reasonable” treewidth
      • Solve using Bucket elimination
    – Else
      • Remove K variables from P’ so that its treewidth is reasonable.
      • z=0
      • For all value assignments $X_k=k$ to the K variables
        – If (F and $X_k=k$) has a solution
          » z=z+Bucket-elimination(P’ | $X_k=k$)
  – Return Z
### BEEM vs VEC on Linkage

<table>
<thead>
<tr>
<th>Problem</th>
<th>W*</th>
<th>N</th>
<th>max K</th>
<th>Space (MB)</th>
<th>BEEM time</th>
<th>VEC time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pedigree1</td>
<td>15</td>
<td>334</td>
<td>4</td>
<td>37</td>
<td>0:00:04</td>
<td></td>
</tr>
<tr>
<td>Pedigree7</td>
<td>30</td>
<td>1068</td>
<td>4</td>
<td>1,607,510</td>
<td>9:50:09</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>Pedigree9</td>
<td>25</td>
<td>1118</td>
<td>7</td>
<td>10,192</td>
<td>0:02:14</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>Pedigree13</td>
<td>31</td>
<td>1077</td>
<td>3</td>
<td>637,157</td>
<td>3:29:35</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>Pedigree18</td>
<td>19</td>
<td>1184</td>
<td>5</td>
<td>166</td>
<td>0:00:06</td>
<td>0:02:15</td>
</tr>
<tr>
<td>Pedigree19</td>
<td>23</td>
<td>793</td>
<td>5</td>
<td>149,928</td>
<td>0:47:53</td>
<td>4:11:40</td>
</tr>
<tr>
<td>Pedigree20</td>
<td>22</td>
<td>437</td>
<td>5</td>
<td>16,510</td>
<td>0:03:42</td>
<td>0:27:46</td>
</tr>
<tr>
<td>Pedigree23</td>
<td>25</td>
<td>402</td>
<td>5</td>
<td>50,301</td>
<td>0:17:30</td>
<td>0:27:47</td>
</tr>
<tr>
<td>Pedigree25</td>
<td>23</td>
<td>1289</td>
<td>5</td>
<td>2,257</td>
<td>0:00:31</td>
<td>0:07:55</td>
</tr>
<tr>
<td>Pedigree30</td>
<td>21</td>
<td>1289</td>
<td>5</td>
<td>563</td>
<td>0:00:12</td>
<td>0:56:30</td>
</tr>
<tr>
<td>Pedigree31</td>
<td>30</td>
<td>1183</td>
<td>5</td>
<td>1,816,694</td>
<td>9:34:54</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>Pedigree33</td>
<td>25</td>
<td>799</td>
<td>4</td>
<td>18,862</td>
<td>0:04:38</td>
<td>10:50:39</td>
</tr>
<tr>
<td>Pedigree34</td>
<td>28</td>
<td>1160</td>
<td>5</td>
<td>657,895</td>
<td>4:43:57</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>Pedigree37</td>
<td>20</td>
<td>1033</td>
<td>5</td>
<td>233,002</td>
<td>1:10:40</td>
<td>3:02:05</td>
</tr>
<tr>
<td>Pedigree38</td>
<td>16</td>
<td>724</td>
<td>5</td>
<td>138,650</td>
<td>0:47:59</td>
<td>0:14:48</td>
</tr>
<tr>
<td>Pedigree39</td>
<td>20</td>
<td>1272</td>
<td>5</td>
<td>900</td>
<td>0:00:15</td>
<td>0:02:54</td>
</tr>
<tr>
<td>Pedigree40</td>
<td>29</td>
<td>1030</td>
<td>7</td>
<td>43,107,736</td>
<td>&gt;24h</td>
<td></td>
</tr>
<tr>
<td>Pedigree41</td>
<td>29</td>
<td>1062</td>
<td>5</td>
<td>881,226</td>
<td>5:00:53</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>Pedigree42</td>
<td>22</td>
<td>448</td>
<td>5</td>
<td>61,149</td>
<td>0:16:04</td>
<td>1:00:08</td>
</tr>
<tr>
<td>Pedigree44</td>
<td>25</td>
<td>811</td>
<td>4</td>
<td>19,187</td>
<td>0:04:42</td>
<td>4:45:22</td>
</tr>
<tr>
<td>Pedigree50</td>
<td>17</td>
<td>514</td>
<td>6</td>
<td>1,465,526</td>
<td>8:55:34</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>Pedigree51</td>
<td>35</td>
<td>1152</td>
<td>5</td>
<td>11,950,172</td>
<td>76:35:16</td>
<td>&gt;48h</td>
</tr>
</tbody>
</table>
## BEEM vs VEC on Linkage

<table>
<thead>
<tr>
<th>Problem</th>
<th>$W^*$</th>
<th>$N$</th>
<th>max K</th>
<th>Space (MB)</th>
<th>BEEM time</th>
<th>VEC time</th>
</tr>
</thead>
<tbody>
<tr>
<td>type 4 100-16</td>
<td>30</td>
<td>6969</td>
<td>5</td>
<td>773,884</td>
<td>4:20:10</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>type 4 100-19</td>
<td>29</td>
<td>7308</td>
<td>5</td>
<td>149,731</td>
<td>0:38:50</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>type 4 120-17</td>
<td>24</td>
<td>7766</td>
<td>5</td>
<td>13,062</td>
<td>0:02:56</td>
<td>3:31:08</td>
</tr>
<tr>
<td>type 4 130-21</td>
<td>29</td>
<td>8844</td>
<td>5</td>
<td>107,434</td>
<td>0:26:12</td>
<td></td>
</tr>
<tr>
<td>type 4 140-19</td>
<td>30</td>
<td>9274</td>
<td>5</td>
<td>847,057</td>
<td>5:04:47</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>type 4 140-20</td>
<td>30</td>
<td>9355</td>
<td>5</td>
<td>2,555,753</td>
<td>6:07:04</td>
<td></td>
</tr>
<tr>
<td>type 4 150-14</td>
<td>32</td>
<td>9449</td>
<td>5</td>
<td>1,051,421</td>
<td>6:16:58</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>type 4 150-15</td>
<td>30</td>
<td>8290</td>
<td>5</td>
<td>2,614,816</td>
<td>15:49:02</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>type 4 160-14</td>
<td>31</td>
<td>10644</td>
<td>5</td>
<td>891,992</td>
<td>4:46:16</td>
<td>&gt;48h</td>
</tr>
<tr>
<td>type 4 160-23</td>
<td>33</td>
<td>13514</td>
<td>5</td>
<td>3,359,257</td>
<td>20:22:31</td>
<td></td>
</tr>
<tr>
<td>type 4 170-23</td>
<td>21</td>
<td>11451</td>
<td>5</td>
<td>567</td>
<td>0:00:36</td>
<td></td>
</tr>
<tr>
<td>type 4 190-20</td>
<td>29</td>
<td>15765</td>
<td>5</td>
<td>389,000</td>
<td>1:58:22</td>
<td>&gt;48h</td>
</tr>
</tbody>
</table>
Conclusions

• BEEM outperforms VEC by a substantial margin on most instances
Exact P(e) : Instances solved

Grids

Linkage 2

Bayes-only solvers

UAI06-PE

Relational
Exact PE : Instances solved overall

- Note: Not weighed by problem class size, biased to some classes/solvers.
Exact $P(e)$: Cumulative time

bn2o

UAI06-MPE

Diagnose

Relational
Summary
(some families show no dominance)

• **Bayes only:**
  – pitt-pe dominates on 3 families
  – Irvine.vec dominates on 1 family

• **Markov only:**
  – ucla-ace-pe dominates on 2 families
  – Irvine.vec dominates on 1 family

• **All networks:**
  – ucla-ace-pe dominates on 2 families
  – Irvine.vec dominates on 2 families