Efficient AND/OR Search Algorithms for Exact MAP Inference Task over Graphical Models

Akihiro Kishimoto
IBM Research, Ireland
Joint work with Radu Marinescu and Adi Botea
Outline

1. Background
2. RBFAOO – Recursive Best-First AND/OR Search with Overestimation
3. SPRBFAOO – Shared-memory Parallelization of RBFAOO
4. Parallel Dovetailing + SPRBFAOO
5. Conclusions
Outline

1. Background
2. RBFAOO – Recursive Best-First AND/OR Search with Overestimation
3. SPRBFAOO – Shared-memory Parallelization of RBFAOO
4. Parallel Dovetailing + SPRBFAOO
5. Conclusions
Graphical Models

- A Graphical Model is a tuple \( \langle X, D, F \rangle \), where
  - \( X = \{X_1, \ldots, X_n\} \) is a set of variables having finite domains \( D = \{D_1, \ldots, D_n\} \).
  - \( F = \{f_1, \ldots, f_r\} \) is a set of positive real-valued local functions.
  - Each function \( f_j \in F \) is defined by \( f_j : Y_j \rightarrow \mathbb{R}^+ \) and \( Y_j \subseteq X \).
  - It represents the function \( C(X) = \prod_{j=1}^{r} f_j(Y_j) \).

- MAP: maximum a posteriori

\[
C^* = C(x^*) = \max_{X} \prod_{j=1}^{r} f_j(Y_j)
\]

- Equivalent with the following energy minimization:

\[
C^* = C(x^*) = \min_{X} \sum_{j=1}^{r} -\log(f_j(Y_j))
\]
Example

\[ C(A, B, C, D, E) = f_1(A, B, C) \times f_2(A, B, D) \times f_3(B, D, E) \]
Solving Exact MAP Inference by AND/OR Tree Search

- Exact MAP inference can be formulated as an AND/OR tree search problem
  - OR Node: Assign one value to one variable
  - AND node: Select one unassigned variable from the set of variables

- The *pseudo tree* determines the set of variables to choose from at each AND node (see the next slide)

- The MAP inference task is converted to the energy minimization task.

- The size of the AND/OR search tree is estimated to be $O(\exp(h))$ where $h$ is the depth of the pseudo tree [Decther & Mateescu, 2004]

- The size of the OR search tree is estimated to be $O(\exp(d))$ where $d$ is the number of variables.
Example: AND/OR Search Tree

- Annotated with edge cost $w(n, m)$ where $n$ is an OR node and $m$ is an AND node.
- Minimize the sum of the edge costs in the solution tree.
- The *optimal solution tree* shown in red indicates an assignment that returns the best value for $C$.
  - Minimization at OR node and summation at AND node
Solving Exact MAP Inference by AND/OR Graph Search

- Two nodes which contain the identical subproblems can be merged into one node. I.e., the search space of exact MAP inference is finite DAG.

- Identical subproblems can be identified by the context that is a partial assignment separating the subproblem from the rest of the problem graph (see next slide) [Decther & Mateescu, 2007]
Example: AND/OR Search Graph

Node $E$ is merged based on the assigned values to variables $B$ and $D$. 
Previous Approaches based on AND/OR Search

- **AND/OR Branch-and-Bound (AOBB)** performs depth-first branch and bound + context-based caching [Marinescu & Dechter, 2009]
  - Pros: Memory efficient as a depth-first search
  - Cons: Explores many subspaces that are irrelevant to optimal solutions

- **Best-First AND/OR search (AOBF)** performs AO*-based best-first search [Marinescu & Dechter, 2009]
  - Pros: Explores the smallest search space
  - Cons: Requires a huge amount of memory

- It is known that many best-first search algorithms are translated into depth-first search algorithms
  - E.g., A* and RBFS/IDA*, proof-number search and depth-first proof-number search, and SSS* and MTD(f)

- Such translation enables to inherit both advantages of depth-first search and best-first search
Contributions

- RBFAOO – A new recursive best-first search algorithm for AND/OR graphs for exact MAP inference
  - Requires limited memory (even linear)
  - Explores nodes in best-first order
- Shared-memory parallelization of RBFAOO
  - Carefully selects a set of nodes to examine in parallel
- Distributed-memory parallelization of RBFAOO based on dovetailing
  - Pass different problem formulations to RBFAOO/SPRBFAOO
Outline

1. Background
2. RBFAOO – Recursive Best-First AND/OR Search with Overestimation
3. SPRBFAOO – Shared-memory Parallelization of RBFAOO
4. Parallel Dovetailing + SPRBFAOO
5. Conclusions
Recursive Best-First AND/OR Search with Overestimation (RBFAOO) 
[Kishimoto & Marinescu, UAI 2014]

Basic Idea

- Transform best-first search (AO* like) into depth-first search using a threshold controlling mechanism (explained next)
  - Based on Korf’s recursive best-first search idea (RBFS) [Korf, 1993] and depth-first proof-number search [Nagai, 2002]
  - Adapted to the context minimal AND/OR graph for graphical models
- Nodes are still expanded in best-first order
- Use admissible function \( h \) that returns a lower-bound of the optimal solution
- Node values called \( q \)-values are updated in the usual manner based on the values of their successors
  - OR - minimization: \( q(n) = \min_{n' \in \text{succ}(n)} (w(n, n') + q(n')) \)
  - AND - summation: \( q(n) = \sum_{n' \in \text{succ}(n)} q(n') \)
  - Initially, \( q(n) = h(n) \) – the heuristic lower bound of the cost below \( n \).
  - \( q(n) \) is improved as the search progresses.
- Some nodes are going to be re-expanded
  - Use caching (limited memory) - done efficiently based on contexts
  - Use overestimation (of the threshold)
Example - step 1

- Expand OR node A by generating its AND successors: \( \langle A, 0 \rangle \) and \( \langle A, 1 \rangle \).
- Best successor is \( \langle A, 0 \rangle \)
- Set threshold \( \theta(A, 0) = 4 \) – indicates next best successor is \( \langle A, 1 \rangle \);
  - we can backtrack to \( \langle A, 1 \rangle \) is the updated cost of the subtree below \( \langle A, 0 \rangle \geq \theta = 4 \).
Example - step 2

- Expand AND node \( \langle A, 0 \rangle \) by generating its OR successors: \( B \) and \( C \).
- Update node value \( q(A, 0) = h(B) + h(C) = 3 \) – threshold OK
• Expand OR node $B$ by generating its AND successor: $\langle B, 0 \rangle$.
• Update node values: $q(B) = 4$ and $q(A, 0) = 6$ – threshold NOT OK
• Backtrack to \( \langle A, 0 \rangle \) and select next best node \( \langle A, 1 \rangle \)
• Set threshold \( \theta(A, 1) = 6 \) (updated value of the left subtree)
• Cache q-value of each expanded node
Some of the nodes in the subtree below \( \langle A, 0 \rangle \) may be re-expanded.
Overestimation

- Simple overestimation scheme for minimizing the node re-expansions
- Inflate the threshold with some small $\delta$: $\theta' = \theta + \delta$ ($\delta > 0$)
  - In practice, we determine $\delta$ experimentally (e.g., $\delta = 1$ worked best)
- Use current best solution cost for bounding
Properties
Correctness and completeness

**Theorem (correctness)**
Given a graphical model $\mathcal{M} = \langle X, D, F \rangle$, if RBFAOO solves $\mathcal{M}$ with admissible heuristic function $h$, its solution is always optimal.

**Theorem (completeness)**
Let $\mathcal{M} = \langle X, D, F \rangle$ be a graphical model with primal graph $G$, let $T$ be a pseudo tree $G$ and let $C_T$ be the context minimal AND/OR search graph based on $T$ (also a finite DAG). Assume that RBFAOO preserves the $q$-values of the nodes $n_1, n_2, \cdots, n_k$ which are on the current search path and the $q$-values of $n_i$’s siblings. Then RBFAOO eventually returns an optimal solution or proves no solution exists.
Benchmarks

• Set of problem instances from the PASCAL2 Inference Challenge
  ▶ pedigree, grid, protein
  ▶ also from http://graphmod.ics.uci.edu/ (Rina Dechter’s group)

• 64 bit C++ code; 2.4GHz 12-core CPU; 80GB of RAM

• Algorithms:
  ▶ AOB(i): Depth-first AND/OR Branch and Bound (full caching)
  ▶ AOBF(i): Best-First AND/OR Search (full caching)
  ▶ RBFAOO(i): Recursive Best-First AND/OR Search with Overestimation
  ▶ All guided by the pre-compiled MBE(i) heuristic
## Results

Genetic linkage analysis

<table>
<thead>
<tr>
<th>instance ((n, k, w^*, h))</th>
<th>algorithm</th>
<th>(i = 6)</th>
<th>(i = 8)</th>
<th>(i = 10)</th>
<th>(i = 12)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>nodes</td>
<td>time</td>
<td>nodes</td>
<td>time</td>
</tr>
<tr>
<td>pedigree7 ((1068,4,28,140))</td>
<td>AOB (\boxed{\text{o}})</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pedigree9 ((1118,7,25,123))</td>
<td>AOB (\boxed{\text{o}})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>1084 195214857</td>
<td>728 136764248</td>
<td>522 97410715</td>
<td>248 46922921</td>
</tr>
<tr>
<td>pedigree13 ((1077,3,30,125))</td>
<td>AOB (\boxed{\text{o}})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pedigree19 ((793,5,21,51))</td>
<td>AOB (\boxed{\text{o}})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pedigree30 ((1289,5,20,105))</td>
<td>AOB (\boxed{\text{o}})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td>83 2648120</td>
<td>103 2689106</td>
<td>45 1717523</td>
<td>24 867988</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>20 5435997</td>
<td>19 5401921</td>
<td>14 3840692</td>
<td>5 1406493</td>
</tr>
<tr>
<td>pedigree39 ((1272,5,20,77))</td>
<td>AOB (\boxed{\text{o}})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td>307 9740964</td>
<td>215 8073776</td>
<td>53 2347928</td>
<td>8 384757</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>79 19804239</td>
<td>67 16260143</td>
<td>14 3461943</td>
<td>2 480866</td>
</tr>
<tr>
<td>pedigree41 ((1062,5,29,119))</td>
<td>AOB (\boxed{\text{o}})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Results for pedigree networks. CPU time (in seconds) and number of nodes expanded. Time limit 1 hour. RBFAOO\((i)\) ran with a 10-20GB cache table (134,217,728 table entries).
## Results

### Binary grid networks

<table>
<thead>
<tr>
<th>instance $(n, k, w^*, h)$</th>
<th>algorithm</th>
<th>$i = 6$</th>
<th>$i = 8$</th>
<th>$i = 10$</th>
<th>$i = 12$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>time</td>
<td>nodes</td>
<td>time</td>
<td>nodes</td>
</tr>
<tr>
<td>50-20-5 (400,2,27,97)</td>
<td>AOB 1163</td>
<td>736</td>
<td>808</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td></td>
<td>AOBF 2148</td>
<td>14256</td>
<td>8080</td>
<td>408</td>
<td>48884418</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>99</td>
<td>385</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td>75-20-5 (400,2,27,99)</td>
<td>AOB 2268</td>
<td>736</td>
<td>8080</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td></td>
<td>AOBF 3076</td>
<td>14256</td>
<td>8080</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>99</td>
<td>385</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td>75-22-5 (484,2,30,107)</td>
<td>AOB 212</td>
<td>643</td>
<td>8080</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td></td>
<td>AOBF 107</td>
<td>14256</td>
<td>8080</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>99</td>
<td>385</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td>75-23-5 (529,2,31,122)</td>
<td>AOB 1860</td>
<td>1109</td>
<td>8080</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td></td>
<td>AOBF 305</td>
<td>14256</td>
<td>8080</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>99</td>
<td>385</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td>90-23-5 (529,2,31,116)</td>
<td>AOB 277</td>
<td>105</td>
<td>8080</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td></td>
<td>AOBF 529</td>
<td>20736</td>
<td>8080</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>99</td>
<td>385</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td>90-26-5 (676,2,36,136)</td>
<td>AOB 1016</td>
<td>1108</td>
<td>8080</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td></td>
<td>AOBF 259</td>
<td>29700</td>
<td>8080</td>
<td>232</td>
<td>48884448</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>99</td>
<td>385</td>
<td>232</td>
<td>48884448</td>
</tr>
</tbody>
</table>

**Table**: Results for grid networks. CPU time (in seconds) and number of nodes expanded. Time limit 1 hour. RBFAOO$(i)$ ran with a 10-20GB cache table (134,217,728 table entries).
## Results

### Protein side chain interaction

<table>
<thead>
<tr>
<th>instance ((n, k, w^*, h))</th>
<th>algorithm</th>
<th>(i = 2)</th>
<th>time</th>
<th>nodes</th>
<th>(i = 3)</th>
<th>time</th>
<th>nodes</th>
<th>(i = 4)</th>
<th>time</th>
<th>nodes</th>
<th>(i = 5)</th>
<th>time</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdb1a3c ((144,81,15,32))</td>
<td>AOBB</td>
<td>-</td>
<td>1915</td>
<td>45513907</td>
<td>oom</td>
<td>344</td>
<td>259261</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td>-</td>
<td>2218</td>
<td>65175805</td>
<td>oom</td>
<td>204</td>
<td>1302</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>8</td>
<td>3195539</td>
<td>6264</td>
<td>8</td>
<td>2694</td>
<td>1492</td>
<td>204</td>
<td>783</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td>pdb1aac ((85,81,11,21))</td>
<td>AOBB</td>
<td>129</td>
<td>2919570</td>
<td>8</td>
<td>2694</td>
<td>oom</td>
<td>oom</td>
<td>204</td>
<td>783</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td>2851</td>
<td>3195539</td>
<td>11</td>
<td>6264</td>
<td>oom</td>
<td>oom</td>
<td>205</td>
<td>3072</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>51</td>
<td>1148212</td>
<td>8</td>
<td>1492</td>
<td>204</td>
<td>783</td>
<td>204</td>
<td>783</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td>pdb1acf ((90,81,9,22))</td>
<td>AOBB</td>
<td>996</td>
<td>55994055</td>
<td>2672</td>
<td>162495198</td>
<td>16</td>
<td>1593</td>
<td>136</td>
<td>4767</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td>22</td>
<td>987416</td>
<td>56</td>
<td>2553896</td>
<td>16</td>
<td>1090</td>
<td>139</td>
<td>10464</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>1148212</td>
<td>1148212</td>
<td>8</td>
<td>1492</td>
<td>204</td>
<td>783</td>
<td>204</td>
<td>783</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td>pdb1ad2 ((177,81,9,33))</td>
<td>AOBB</td>
<td>259</td>
<td>7770890</td>
<td>134</td>
<td>3806154</td>
<td>657</td>
<td>3312980</td>
<td>2552</td>
<td>274955</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td>831</td>
<td>1250161</td>
<td>394</td>
<td>715399</td>
<td>1109</td>
<td>1049637</td>
<td>2595</td>
<td>135265</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>36</td>
<td>1227741</td>
<td>42</td>
<td>858899</td>
<td>585</td>
<td>1218780</td>
<td>2543</td>
<td>113780</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td>pdb1ail ((62,81,8,23))</td>
<td>AOBB</td>
<td>4</td>
<td>177150</td>
<td>31</td>
<td>75051</td>
<td>610</td>
<td>1474224</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td>80</td>
<td>66207</td>
<td>47</td>
<td>15817</td>
<td>1728</td>
<td>928375</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>2</td>
<td>78677</td>
<td>30</td>
<td>16427</td>
<td>599</td>
<td>1311325</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td>pdb1atg ((175,81,12,39))</td>
<td>AOBB</td>
<td>-</td>
<td>2347033</td>
<td>6</td>
<td>154434</td>
<td>260</td>
<td>9348036</td>
<td>236</td>
<td>24412</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>AOBF</td>
<td>38</td>
<td>119195</td>
<td>632</td>
<td>1196429</td>
<td>247</td>
<td>30072</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
<tr>
<td></td>
<td>RBFAOO</td>
<td>620</td>
<td>24347033</td>
<td>4</td>
<td>71446</td>
<td>32</td>
<td>430747</td>
<td>236</td>
<td>11545</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
<td>oom</td>
</tr>
</tbody>
</table>

**Table:** Results for protein networks. CPU time (in seconds) and number of nodes expanded. Time limit 1 hour. RBFAOO\((i)\) ran with a 10-20GB cache table (134,217,728 entries).
Results

Figure: Normalized total CPU time as a function of the $i$-bound.
Impact of overestimation

Figure: Node re-expansion rate and percentage of instances solved by RBFAOO(i) as a function of the overestimation rate $\delta$. Time limit 1 hour. $i = 10$ for grids and pedigrees, $i = 4$ for protein.
Outline

1. Background
2. RBFAOO – Recursive Best-First AND/OR Search with Overestimation
3. SPRBFAOO – Shared-memory Parallelization of RBFAOO
4. Parallel Dovetailing + SPRBFAOO
5. Conclusions
Shared-memory Parallel RBFAOO (SPRBFAOO)  
[Kishimoto, Marinescu & Botea, NIPS 2015]  

Obstacles to Efficient Parallelization

- Shared-memory parallel search suffers from search and synchronization overheads.
  - Search overhead refers to extra states examined by parallel search
  - Synchronization overhead is the idle time wasted at the synchronization points including locks for mutual exclusion.

- These overheads are usually inter-dependent. 
  It is difficult to theoretically find the right mix of these overhead.

- RBFAOO has a characteristic of best-first search that tries to examine only the promising portions of the search space.
  I.e., RBFAOO attempts not to examine useless portions of the search space.

- How can we initiate parallelism without increasing the search overhead?
Shared-memory Parallel RBFAOO (SPRBFAOO)

Basic Idea

- Use ideas behind parallel depth-first proof-number search [Kaneko, 2010]
- All threads start from the root with the same search and with one shared cache table.
- The virtual q-value $vq(n)$ for node $n$ is used to control parallel search.
  1. Initially, $vq(n)$ is set to $q(n)$.
  2. When a thread examines $n$, $vq(n)$ is incremented by a small value $\zeta$.
  3. When all the threads finish examining $n$, $vq(n)$ is set to $q(n)$.
- An effective load balancing is obtained without any sophisticated schemes, while promising portions of the search space are examined.
Example – Step 1

- Using 2 threads and $\eta = 1$. 

![Diagram showing tree structure with nodes A, B, C, D, and edge labels]

Thread 1 at A
- th(B)=3
- q(B)=2, vq(B)=2, q(C)=3, vq(C)=3
- q(D)=10, vq(D)=10

Thread 2 at A
Example – Step 2

- Using 2 threads and $\eta = 1$.

```
Thread 1 at B
th(B)=3
q(B)=2, vq(B)=3

Thread 2 at A
th(C)=3
q(C)=3, vq(C)=3
```
Example – Step 3

- Using 2 threads and $\eta = 1$.

```
+---A---+
 |   1   |
 |       |
+---B---+---C---+---D---+
 |   0   |   0   |   0   |
 |       |       |       |
+---E---+---F---+---G---+
```

Thread 1 at B

- $th(B)=3$
- $q(B)=2$, $vq(B)=3$

Thread 2 at C

- $th(C)=3$
- $q(C)=3$, $vq(C)=4$
Example – Step 4

- Using 2 threads and $\eta = 1$. 

Thread 1 at B
- th(B)=3
- q(B)=2, vq(B)=3, q(C)=6, vq(C)=6

Thread 2 at A
Example – Step 5

- Using 2 threads and $\eta = 1$. 

Thread 1 at E
- $\text{th}(B)=3$
- $\text{th}(E)=2$

Thread 2 at B
- $\text{th}(B)=6$

$q(B)=2$, $vq(B)=4$, $q(C)=6$, $vq(C)=6$
$q(E)=1$, $vq(E)=2$
Properties
Correctness

**Theorem**

*Given a graphical model $\mathcal{M} = \langle X, D, F \rangle$, and an admissible heuristic $h$, SPRBFAOO returns optimal solutions.*
Benchmarks

- The same set of problem instances used in RBFAOO experiments
  - pedigree, grid, protein
  - also from http://graphmod.ics.uci.edu/ (Rina Dechter's group)
- 64 bit C++ code; 2.4GHz 12-core CPU; 80GB of RAM
Results
Small $i$-bounds, 12 threads, $\eta = 0.01$
Results
Large $i$-bounds, 12 threads, $\eta = 0.01$

Total CPU time (sec) for RBFAOO vs. SPRBFAOO (time limit 2 hours)
Outline

1. Background
2. RBFAOO – Recursive Best-First AND/OR Search with Overestimation
3. SPRBFAOO – Shared-memory Parallelization of RBFAOO
4. Parallel Dovetailing + SPRBFAOO
5. Conclusions
Dovetailing-based Parallel RBFAOO
[Marinescu, Kishimoto & Botea, ECAI 2016]

- Parallel dovetailing is a simple way of parallelization when the best parameter to start with is not known beforehand.
  - Given a graphical model \( M \) and \( m \) processing cores, run in parallel \( m \) instances of an exact MAP algorithm \( a \).
  - As a different parameter configuration \( \theta_i \) (\( 1 \leq i \leq m \)), use the pseudo tree generated by a randomized \textit{min-fill} heuristic [Kjaerulff 1990].
  - When a solution is found for one of the \( m \) instances, that solution is proven to be optimal.
  - The performance of AND/OR search is strongly influenced by the quality of the pseudo tree that determines variable ordering [Marinescu & Dechter 2009].
- This approach often works efficiently in distributed-memory environments.
  - In addition to search and synchronization overheads, distributed-memory parallel search that attempts to partition search spaces suffers from communication overhead.
  - Parallel dovetailing incurs almost no communication overhead.
Baseline SPRBFAOO runs on 12 cores guided by the pre-compiled MBE(i) heuristic ($i = 6, 14$ for pedigree and grid, $i = \{2, 4\}$ for protein).
Outline

1. Background
2. RBFAOO – Recursive Best-First AND/OR Search with Overestimation
3. SPRBFAOO – Shared-memory Parallelization of RBFAOO
4. Parallel Dovetailing + SPRBFAOO
5. Conclusions
Conclusions

- RBFAOO – limited memory best-first AND/OR search for graphical models, thus bridging the gap between depth-first and best-first search methods.

- SPRBFAOO – parallelization of RBFAOO that leads to considerable speedups (up to 7-fold using 12 threads) especially on hard problem instances.

- Parallel Dovetailing + SPRBFAOO – combination that further improves RBFAOO’s performance.

- Empirical evaluation on a set of difficult pedigree networks that demonstrates clearly the benefit of these approaches.

- **Future work:**
  - Extend it into an anytime scheme
  - Exploitation of better pseudo tree
  - Parallelization based on partitioning search spaces in distributed-memory environments
  - Parallelization in heuristic construction