Look-Ahead with Mini-Bucket Heuristics for MPE

Rina Dechter\textsuperscript{1}, Kalev Kask\textsuperscript{1}, \textbf{William Lam}\textsuperscript{1}, Javier Larrosa\textsuperscript{2}

\textsuperscript{1}University of California, Irvine
\textsuperscript{2}UPC Barcelona Tech
Outline

- Background
  - Graphical models, MPE, Branch-and-Bound
  - Look-ahead
  - Mini-bucket Heuristic

- Bucket Error

- Look-ahead with Bucket Error

- Experiments

- Conclusions
Graphical Models and Finding an Optimal Assignment
[Marinescu and Dechter 2008]

\[ f_1(A) + f_2(A, B) + f_3(A, D) + f_4(A, G) + f_5(B, C) + f_6(B, D) + f_7(B, E) + f_8(B, F) + f_9(C, D) + f_{10}(C, E) \]

\[ F(X) = \sum_{i=1}^{10} f_i(X) \]

\[ MPE = \min_{X} F(X) \]

**Approach: Search**
- Depth-first AND/OR Branch and Bound (AOBB)
Depth-First Branch and Bound (DFBB)

Each node $n$ is a **subproblem** (defined by current conditioning)

- $g(n) =$ cost to get to node $n$
- $f(n) = g(n) + h(n)$ estimate of overall solution given the conditioning to node $n$
- $h(n) =$ lower bound of the best solution in the sub-tree

If $f(n)$ is worse than current best solution, then prune.
Look-Ahead in Search

- Given that $s_1, ..., s_t$ are child nodes of $s$ in the search space and $w(s, s_i)$ is the weight of the arc from $s$ to $s_i$, $h_{lh}^{d}(s)$ be the $d$-level lookahead function of $s$, then

$$h_{lh}^{1}(s) = \min_{\{s_1, ..., s_t\} \text{ in child}(s)} \{w(s, s_i) + h(s_i)\}$$

$$h_{lh}^{d}(s) = \min_{\{s_1, ..., s_t\} \text{ in child}(s)} \{w(s, s_i) + h_{lh}^{d-1}(s_i)\}$$

- The (1-level) residual: $res_h(s) = h_{lh}^{1}(s) - h(s)$
- Can be viewed as a search problem over the next $d$ levels

- **Our focus**: Can we cost-effectively improve our heuristic with look-ahead?
Bucket and Mini-Bucket Elimination

\[ f_1(A) + f_2(A, B) + f_3(A, D) + f_4(A, G) + f_5(B, C) + f_6(B, D) + f_7(B, E) + f_8(B, F) + f_9(C, D) + f_{10}(C, E) \]

**Bucket Elimination (BE) (Dechter 1999)**
- Solves the min-sum problem by eliminating variables one at a time.
- Complexity: exponential in the \( w^* \) of the underlying primal graph

**Mini-Bucket Elimination (MBE) (Dechter and Rish 2001)**
- We can approximate BE by solving a relaxation created by duplicating variables
- to bound the \( w^* \) by a parameter known as the \( i \)-bound.

\[
F(X) = \sum_{i=1}^{10} f_i(X)
\]

\[
MPE = \min_X F(X)
\]
Bucket and Mini-Bucket Elimination

\[ f_1(A) + f_2(A, B) + f_3(A, D) + f_4(A, G) + f_5(B, C) + f_6(B, D) + f_7(B, E) + f_8(B, F) + f_9(C, D) + f_{10}(C, E) \]

Bucket Elimination (BE) (Dechter 1999)
- Solves the min-sum problem by eliminating variables one at a time.
- Complexity: exponential in the \( w^* \) of the underlying primal graph.

Mini-Bucket Elimination (MBE) (Dechter and Rish 2001)
- We can approximate BE by solving a relaxation created by duplicating variables to bound the \( w^* \) by a parameter known as the \( i \)-bound.

\[ \lambda^D(A, B, C) \geq \lambda^D(A) + \lambda^D(B, C) \]

- **Bucket Error**
  \[ \text{Err}_D(A, B, C) = \lambda^D(A, B, C) - (\lambda^D(A) + \lambda^D(B, C)) \]
  - Captures the local error of the bucket.
  - When a bucket is not partitioned, Err is trivially zero.
Mini-Bucket Errors and the i-bound

i-bound=3

Err_D(A,B,C) = λ^D(A,B,C) - (λ^D(A) + λ^D(B) + λ^D(C))

Err_E(B,C) = λ^E(B,C) - (λ^E(B) + λ^E(C))

Err_G(A,F) = λ^G(A,F) - (λ^G(A) + λ^G(F))

i-bound=2

Err_D(A,B,C) = λ^D(A,B,C) - (λ^D(A) + λ^D(B) + λ^D(C))

Err_E(B,C) = λ^E(B,C) - (λ^E(B) + λ^E(C))

Err_G(A,F) = λ^G(A,F) - (λ^G(A) + λ^G(F))
Bucket Error Evaluation (BEE)

**Algorithm 1: Bucket Error Evaluation (BEE)**

**Input:** A Graphical model $\mathcal{M} = (X, D, F)$, a pseudo-tree $T$, $i$-bound

**Output:** The error function $Err_k$ for each bucket

**Initialization:** Run $MBE(i)$ w.r.t. $T$.

for each $B_k, X_k \in X$ do

Let $B_k = \bigcup_r B^r_k$ be the partition used by $MBE(i)$

1. $\mu_k = \sum_{r} (\min X_k \sum_{f \in B^r_k} f)$
2. $\mu^*_k = \min X_k \sum_{r} f_{B_k}$
3. $Err_k \leftarrow \mu^*_k - \mu_k$

return $Err$ functions
Complexity of BEE

**Theorem 1 (Complexity of BEE)**
The complexity of BEE is $O(n \cdot k^{pw(i)})$, where $n$ is the number of variables, $k$ bounds the domain size and $pw(i)$ is the pseudo-width along $T$ relative to MBE(i).

$psw_j$: the pseudo-width of bucket $j$ is the number of variables in the bucket at the time of processing.
Residual and Bucket Error

**Theorem 2 (residual and bucket-error)** Assume an execution of $MBE(i)$ along $T$ yielding heuristic $h$. Then, for every $x_p$

$$res(x_p) = \sum_{X_k \in ch(X_p)} Err_k(x_p)$$

(9)
**Minimal Look-ahead Subtree**

**Definition 6 (average relative bucket error)** The average relative bucket error of $X_j$ given a run of MBE(i) is

$$\tilde{E}_j = \frac{1}{|\text{dom}(B_j)|} \sum_{\bar{x}_j} \frac{Err_j(\bar{x}_j)}{\mu^*(\bar{x}_j)}$$

Look-ahead subtrees from A, by depth

Pseudo-tree: Red nodes are look-ahead *relevant* (relative error above a certain threshold)
Sample of a part of a pseudodotree (pedigree40)

pedigree40: n=842, w=27, h=111, k=7, ib=6

depth = 5,
threshold = 0
Sample of a part of a pseudotree (pedigree40)

pedigree40: n=842, w=27, h=111, k=7, ib=6

depth = 3, threshold = 0
Sample of a part of a pseudotree (pedigree40)

pedigree40: n=842, w=27, h=111, k=7, ib=6

depth = 5,
threshold = 0
Sample of a part of a pseudotree (pedigree40)

pedigree40: n=842, w=27, h=111, k=7, ib=6

depth ≥ 3, threshold = 9.5
Bucket-Errors: Across i-bounds (pedigree40)

pedigree40: n=842, w=27, h=111, k=7, ib=6

pedigree40: n=842, w=27, h=111, k=7, ib=10

pedigree40: n=842, w=27, h=111, k=7, ib=14
Experiments

- AOBB on the context-minimal AND/OR graph
- MBE-MM heuristic with different $i$-bounds
- Time limit: 6 hours
- MBE memory: 4GB
- Pruned look-ahead trees with BEE
  - Compute error functions based on sampling at most $10^5$ entries. (Exact if there are fewer entries.)
  - Error threshold of 0.01.
Experiments

- **Benchmark Statistics**

| Benchmark   | # inst | n     | k  | w  | h  | |F| | a |
|-------------|--------|-------|----|----|----|---|---|
| Pedigree    | 17     | 387   | 3  | 19 | 58 | 438| 4 |
|             |        | 1015  | 7  | 39 | 143| 1290| 5 |
| LargeFam    | 14     | 950   | 3  | 32 | 66 | 1383| 4 |
|             |        | 1530  | 3  | 40 | 95 | 2352| 4 |
| Promedas    | 28     | 735   | 2  | 41 | 77 | 749| 3 |
|             |        | 2113  | 2  | 120| 180| 2134| 3 |
# Experiments

<table>
<thead>
<tr>
<th>instance</th>
<th>Lookahead depth</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i=5</td>
<td>i=8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pedigree</td>
<td>none</td>
<td>1262</td>
<td>826K</td>
<td>35</td>
<td>23K</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>912</td>
<td>564K</td>
<td>20</td>
<td>13K</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>691</td>
<td>311K</td>
<td>12</td>
<td>6K</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>300</td>
<td>66K</td>
<td>13</td>
<td>2K</td>
</tr>
<tr>
<td></td>
<td>i=17</td>
<td>i=18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lf3_11_53</td>
<td>none</td>
<td>1042</td>
<td>6730K</td>
<td>4349</td>
<td>2809K</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>8611</td>
<td>4875K</td>
<td>3653</td>
<td>2116K</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5481</td>
<td>1674K</td>
<td>2750</td>
<td>901K</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2014</td>
<td>583K</td>
<td>1091</td>
<td>323K</td>
</tr>
</tbody>
</table>
Experiments: Summary over Instances

Figure 2: Total CPU times for using no look-ahead plotted against best total CPU times for any level of look-ahead (in log scale). Each point in the gray region represents an instance where the look-ahead heuristic has better time performance for some setting of the depth. We also report the number of instances which were actually solved and the average speedup across these instances.
Experiments: Summary over Instances

Pedigree networks (medium i-bounds)

LargeFam linkage networks (medium i-bounds)

Promedas networks (medium i-bounds)
Conclusion

- We introduced the notion of bucket error to estimate the accuracy of the MBE heuristic.
- We can make look-ahead cost-effective for MBE heuristics using bucket errors.
- Future work: Applying the techniques described here to best-first search and work towards anytime algorithms that produce lower and upper bounds.
Thanks!