Chapter 10
Hybrid of search and inference
Time-space tradeoff
Outline

- Acyclic networks
- Join-tree clustering
- Conditioning vs tree-clustering
Transforming into a tree

- **By Inference**
  - Time and space exponential in tree-width

- **By Conditioning-search**
  - Time exponential in the cycle-cutset
Treewidth equals cycle cutset

$\text{treewidth} = \text{cycle cutset} = 4$
Treewidth smaller than cycle cutset

treewidth = 2

cycle cutset = 5
DR versus DPLL: Complementary Properties

Uniform random 3-CNFs (large induced width)

(k,m)-tree 3-CNFs (bounded induced width)
## Complementary behavior of search vs variable elimination

<table>
<thead>
<tr>
<th></th>
<th>Search</th>
<th>Variable Elimination</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Worst-case time</strong></td>
<td>Better than worst-case</td>
<td>Same as worst-case</td>
</tr>
<tr>
<td><strong>Average time</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Space</strong></td>
<td>One solution</td>
<td>Knowledge compilation</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How to combine search and inference

- *Pre-processing* by bounded inference and then search
- *Alternate search and inference* (do inference on a subset of variables, resulting in a smaller problem on which we can search)
- Search with look-ahead (e.g. arc-consistency) doing inference at every node in the search tree.
- We will illustrate 2 general architectures for hybrids.
 Conditioning
 Specialized cutset schemes

- Inference may require too much memory
- **Condition** on some of the variables
Conditioning

Graph Coloring problem

- Inference may require too much memory
- **Condition** on some of the variables
Cycle cutset

Cycle cutset = \{A, B, C\}
The Cycle-Cutset Effect

- A cycle-cutset is a subset of nodes in an undirected graph whose removal results in a graph with no cycles
- An instantiated variable cuts flow of information cycles
- If a cycle-cutset is instantiated the remaining problem is a tree and can be solved efficiently

![Diagram](image)
Example of the cycle-cutset scheme
Theorem 10.1.2 Algorithm cycle-cutset decomposition has time complexity of $O(nk^{c+2})$
where $n$ is the number of variables, $c$ is the cycle-cutset size and $k$ is the domain size.
The space complexity of the algorithm is linear.
From cycle-cutset to b-cutset (or w-cutset)

**Definition 10.2.1 (b-cutset)** Given a graph $G$, a subset of nodes is called a b-cutset iff when the subset is removed the resulting graph has an induced-width less than or equal to $b$. A minimal b-cutset of a graph has a smallest size among all b-cutsets of the graph. A cycle-cutset is a 1-cutset of a graph.

How can we find a b-cutset? A minimal one? Let’s use a variable ordering and remove one variable after another until we get a b-cutset.
Adjusted Induced-width

It is clear that finding a minimal $b$-cutset is a hard task, though we can define a $b$-cutset relative to the variable ordering. Given an ordering $d = x_1, ..., x_n$ of $G$, a $b$-cutset relative to $d$ is obtained by processing the nodes from last to first. When node $x$ is processed, if its induced-width is greater than $b$ it is added to the $b$-cutset. Otherwise, its earlier neighbors are connected. The induced-width relative to a cutset is called \textit{adjusted induced-width}. The adjusted induced-width relative to a $b$-cutset is $b$. Clearly a minimal $b$-cutset is a smallest among all $b$-cutsets.
Algorithm elim-cond(b)

Input: A constraint network \( \mathcal{R} = (X, D, C) \), \( Y \subseteq X \) which is a b-cutset. \( d \) is an ordering that starts with \( Y \) such that the adjusted induced-width, relative to \( Y \) along \( d \), is bounded by \( b \), \( Z = X - Y \).

Output: A consistent assignment, if there is one.

1. while \( \bar{y} \leftarrow \) next partial solution of \( Y \) found by backtracking, do
   
   (a) \( \bar{z} \leftarrow \text{adaptive-consistency}(\mathcal{R}_{Y=\bar{y}}) \).
   
   (b) if \( \bar{z} \) is not false, return solution = \((\bar{y}, \bar{z})\).

2. endwhile.

3. return: the problem has no solutions.

Figure 10.5: Algorithm elim-cond(b)
Time-space tradeoff

**Theorem:**

The b-cutset scheme yields space complexity $O(k^b)$ and time complexity $O(n k^{b+ c_b})$, where $c_b$ is the size of the b-cutset.

As $b$ decreases, $c_b$ increases.

The cycle-cutset decomposition is linear space and has time complexity of
Space-time tradeoffs

In general:

\[ 1 + c_1 \geq 2 + c_2 \geq \ldots b + c_b, \ldots \geq w^* + c_{w^*} = w^* \]

\( C_i \) is the smallest i-cutset

- Space complexity \( O(k^w) \)
- Time complexity \( O(n k^{b+ c_w}) \), where \( c_w \) is the size of the w-cutset.
- As \( w \) decreases, \( c_w \) increases.
Hybrid algorithms for propositional theories
W-cutset Example

\((\neg C \vee E)(A \vee B \vee C \vee D)(\neg A \vee B \vee E \vee D)(B \vee C \vee D)\)
Review: Cluster Tree Elimination

• Cluster Tree Elimination (CTE) works by passing messages along a tree-decomposition

• Basic idea:
  • Each node sends one message to each of its neighbors
  • Node $u$ sends a message to its neighbor $v$ only when $u$ received messages from all its other neighbors
Constraint propagation
Join-Tree Decomposition
(Dechter and Pearl 1989)

- Each function in a cluster
- Satisfy running intersection property
- Tree-width: number of variables in a cluster-1
- Equals induced-width
The bottom up messages

Bucket G: \( R(G, F) \)

Bucket F: \( R(F, B, C) \) \( \rightarrow \rho^F_G(F) \) \( \rho^F_C(B, C) \)

Bucket D: \( R(D, A, B) \) \( \rightarrow \rho^D_C(B, C) \)

Bucket C: \( R(C, A) \) \( \rightarrow \rho^C_F(B, C) \)

Bucket B: \( R(B, A) \) \( \rightarrow \rho^B_D(A, B) \) \( \rho^B_C(A, B) \) \( \rho^B_A(A) \)

Bucket A: \( R(A) \) \( \rho^A_B(A) \)
Theorem 10.3.1 Given a tree-decomposition $T$ over $n$ variables, separator sizes $s_0; s_1, \ldots, s_t$ and secondary tree-decompositions having a corresponding maximal number of nodes in any cluster, $r_0, r_1, \ldots, r_t$. The complexity of CTE when applied to each secondary tree-decompositions $T_i$ is $O(n \exp(r_i))$ time, and $O(n \exp(s_i))$ space ($i$ ranges over all the secondary tree-decomposition).

Figure 10.12: A tree-decomposition with separators equal to (a) 3, (b) 2, and (c) 1
Super-buckets

Figure 10.13: From a bucket-tree (left) to join-tree (middle) to a super-bucket-tree (right)
The idea of super-buckets/clusters

Larger super-buckets (cliques) => more time but less space

Complexity:
1. Time: exponential in clique (super-bucket) size
2. Space: exponential in separator size

Algorithm CTE(b)” generate a tree-decomposition with separator size bound by b. Then, apply algorithm CTE.
Hybrid of Hybrids.

Algorithm hybrid\((b_1; b_2)\): Apply elim-cond\((b_2)\) inside each cluster of a b-tree-decomposition.
Space complexity \(\exp(b_1)\) time complexity \(\exp(b_2)\), \(b_2 \leq b_1\).

Hybrid\((b_1,1)\): apply cycle-cutset in each cluster (see circuit examples next)

Hybrid\((1,1)\): apply cycle-cutset for each non-separable component.
Application: circuit diagnosis

**Problem:** Given a circuit and its unexpected output, identify faulty components. The problem can be modeled as a constraint optimization problem and solved by bucket elimination.
Figure 10.17: Primary join tree (157 cliques) for circuit c432 (196 variables); the maximum separator size is 23.
Figure 10.20: Secondary trees for c432 with separator sizes 16 and 11, 7 and 3.
Figure 10.19: Time/Space tradeoff for c432 (196 variables), c499 (243 variables), c880 (443 variables), c1355 (587 variables), c1908 (913 variables) and c2670 (1426 variables). Time is measured by the maximum of the separator size and the cutset size and space by the maximum separator size.
Constraint Optimization and counting
Chapter 13
Outline

• **Introduction**
  - Optimization tasks for graphical models
  - Solving optimization problems with inference and search

• **Inference**
  - Bucket elimination, dynamic programming
  - Mini-bucket elimination

• **Search**
  - Branch and bound and best-first
  - Lower-bounding heuristics
  - AND/OR search spaces
Outline

• Introduction
  • Optimization tasks for graphical models
  • Solving optimization problems with inference and search

• Inference
  • Bucket elimination, dynamic programming
  • Mini-bucket elimination

• Search
  • Branch and bound and best-first
  • Lower-bounding heuristics
  • AND/OR search spaces
Constraint optimization problems for graphical models

f(A,B,D) has scope \(\{A, B, D\}\)
Constraint Optimization Problems for Graphical Models

Primal graph =
Variables --> nodes
Functions, Constraints --> arcs

\[ F(A,B,C,D,F,G) = f_1(A,B,D) + f_2(D,F,G) + f_3(B,C,F) \]

\[ f(A,B,D) \text{ has scope } \{A,B,D\} \]
Example: constrained optimization

Example: power plant scheduling

```
<table>
<thead>
<tr>
<th>Unit #</th>
<th>Min Up Time</th>
<th>Min Down Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Example: combinatorial auction

Given a set of elements \( A = \{a_1, \ldots, a_n\} \) and given a set of bids, \( B = \{b_1, \ldots, b_l\} \), each is a subset of \( A \), each having cost \( r_i \), select a subset of bids \( B' \) with maximal total cost where no two bids share an item.

\[
\max_{B'} \sum_{b_i \text{ in } B'} r_i
\]

Consider a problem instance given by the following bids: \( b_1 = \{1, 2, 3, 4, \} \), \( b_2 = \{2, 3, 6\} \), \( b_3 = \{1, 5, 4\} \), \( b_4 = \{2, 8\} \), \( b_5 = \{5, 6\} \) and the costs \( r_1 = 8 \), \( r_2 = 6 \), \( r_3 = 5 \), \( r_4 = 2 \), \( r_5 = 2 \). In this case the variables are \( b_1, b_2, b_3, b_4, b_5 \), their domains are \( \{0, 1\} \) and the constraints are \( R_{12}, R_{13}, R_{14}, R_{24}, R_{25}, R_{35} \). The cost network for this problem formulation is identical to its constraint network, since all the cost components are unary. The reader can verify that an optimal solution is given by \( b_1 = 0, b_2 = 1, b_3 = 1, b_4 = 0, b_5 = 0 \). Namely, selecting bids \( b_2 \) and \( b_3 \) is an optimal choice with total cost of 11.
Probabilistic Networks

\[ P(S, C, B, X, D) = P(S) \cdot P(C|S) \cdot P(B|S) \cdot P(X|C, S) \cdot P(D|C, B) \]

MPE: Find a maximum probability assignment, given evidence

\[ \text{Find } \arg \max P(S) \cdot P(C|S) \cdot P(B|S) \cdot P(X|C, S) \cdot P(D|C, B) \]
Graphical models

- A graphical model \((X,D,F)\):
  - \(X = \{X_1,\ldots,X_n\}\) variables
  - \(D = \{D_1, \ldots, D_n\}\) domains
  - \(F = \{f_1,\ldots,f_m\}\) functions

- Operators:
  - combination
  - elimination (projection)

- Tasks:
  - Belief updating: \(\sum_{x,y} \prod_j p_j\)
  - MPE: \(\max_x \prod_j p_j\)
  - CSP: \(\prod_x \times_j c_j\)
  - Max-CSP: \(\min_x \sum_j f_j\)

- All these tasks are NP-hard
  - exploit problem structure
  - identify special cases
  - approximate
### Combination of cost functions

#### $f(A,B)$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$f(A,B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td>6</td>
</tr>
<tr>
<td>b</td>
<td>g</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>g</td>
<td>6</td>
</tr>
</tbody>
</table>


#### $f(B,C)$

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>$f(B,C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td>6</td>
</tr>
<tr>
<td>b</td>
<td>g</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>g</td>
<td>6</td>
</tr>
</tbody>
</table>


#### $f(A,B,C)$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>$f(A,B,C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
<td>12</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>g</td>
<td>6</td>
</tr>
<tr>
<td>b</td>
<td>g</td>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>g</td>
<td>g</td>
<td>6</td>
</tr>
<tr>
<td>g</td>
<td>b</td>
<td>b</td>
<td>6</td>
</tr>
<tr>
<td>g</td>
<td>b</td>
<td>g</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>g</td>
<td>b</td>
<td>6</td>
</tr>
<tr>
<td>g</td>
<td>g</td>
<td>g</td>
<td>12</td>
</tr>
</tbody>
</table>


#### Combination

$$f(A,B) + f(B,C) = 0 + 6$$

Fall 2022
Outline

• Introduction
  • Optimization tasks for graphical models
  • Solving by inference and search

• Inference
  • Bucket elimination, dynamic programming, tree-clustering, bucket-elimination
  • Mini-bucket elimination, belief propagation

• Search
  • Branch and bound and best-first
  • Lower-bounding heuristics
  • AND/OR search spaces

• Hybrids of search and inference
  • Cutset decomposition
  • Super-bucket scheme
Conditioning vs. Elimination

Conditioning (search)

Elimination (inference)

A=1

A=k

k “sparser” problems

1 “denser” problem
Computing the optimal cost solution

Constraint graph

\[ \text{OPT} = f(a,b) + f(a,c) + f(a,d) + f(b,c) + f(b,d) + f(b,e) + f(c,e) \]

Variable Elimination

Combination
Algorithm **elim-opt** (Dechter, 1996)
Non-serial Dynamic Programming (Bertele and Briochi, 1973)
Generating the optimal assignment

B: \( f(a,b) \) \( f(b,c) \) \( f(b,d) \) \( f(b,e) \)

C: \( f(c,a) \) \( f(c,e) \)

D: \( f(a,d) \)

E: \( e=0 \)

A:
Complexity

Algorithm **elim-opt** (Dechter, 1996)
Non-serial Dynamic Programming (Bertele and Briochi, 1973)

- **Bucket B**: $f(a,b)$, $f(b,c)$, $f(b,d)$, $f(b,e)$
- **Bucket C**: $f(c,a)$, $f(c,e)$
- **Bucket D**: $f(a,d)$
- **Bucket E**: $e=0$
- **Bucket A**: $\exp(\text{w}^*=4)$

"induced width" (max clique size)
Induced width

Bucket-elimination is time and space

Finding smallest induced-width is hard
Using a different ordering

Figure 13.1: The cost graph of the cost function: \( C(a, b, c, d, f, g) = F_0(a) + F_1(a, b) + F_2(a, c) + F_3(b, c, f) + F_4(a, b, d) + F_5(f, g) \)

Bucket G: \( F_5(f, g) \)
Bucket D: \( F_4(a, b, d) \)
Bucket F: \( F_3(b, c, f) \)
Bucket B: \( F_1(b, a) \)
Bucket C: \( F_2(c, a) \)
Bucket A: \( F_0(a) \)

(a)

max \( \sum \)

Bucket G: \( F_5(f, g) \)
Bucket D: \( F_4(a, b, d) \)
Bucket F: \( F_3(b, c, f) \)
  \( h^G(f) \)
Bucket B: \( F_1(b, a) \)
  \( h^B(a, b) \)
  \( h^F(b, c) \)
Bucket C: \( F_2(c, a) \)
  \( h^B(a, c) \)
Bucket A: \( F_0(a) \)
  \( h^C(a) \)

(b)

Figure 13.4: Bucket elimination along ordering \( d_1 = A, C, B, F, D, G \).
Induced-width (again...)

Figure 13.7: Two orderings of the cost graph of our example problem
### Algorithm elim-opt

**Input:** A cost network $C = (X, D, C)$, $C = \{F_1, \ldots, F_l\}$; ordering $d$

**Output:** The maximal cost assignment to $\sum_j F_j$.

1. **Initialize:** Partition the cost components into ordered buckets.
2. **Process buckets** from $p \leftarrow n$ downto 1
   For costs $h_1, h_2, \ldots, h_j$ defined over scopes $Q_1, \ldots, Q_j$ in $\text{bucket}_p$, do:
   - **If** (observed variable) $x_p = a_p$, assign $x_p = a_p$ to each $h_i$ and put in appropriate buckets. Terminate if value is inconsistent.
   - **Else,** (sum and maximize)
     
     $A \leftarrow \bigcup_i Q_i - \{x_p\}$
     
     $h^p = \max_{x_p} \sum_{i=1}^j h_i$.
     
     Place $h^p$ in the latest lower bucket mentioning a variable in $A$.

3. **Forward:** From $i = 1$ to $n$, given $\vec{a}_{i-1}$, assign $x_i$ a value $a_i$ that maximizes the sum values of functions in its bucket.

---

**Figure 13.5:** Dynamic programming as elim-opt

---

**Fall 2022**
Example: combinatorial auction

Given a set of elements $A = \{a_1, \ldots, a_n\}$ and given a set of bids, $B = \{b_1, \ldots, b_l\}$, each is a subset of $A$, each having cost $r_i$, select a subset of bids $B'$ with maximal total cost where no two bids share an item.

$$\max_{B'} \max \sum_{b_i \text{ in } B'} r_i$$

Consider a problem instance given by the following bids: $b_1 = \{1, 2, 3, 4\}$, $b_2 = \{2, 3, 6\}$, $b_3 = \{1, 5, 4\}$, $b_4 = \{2, 8\}$, $b_5 = \{5, 6\}$ and the costs $r_1 = 8$, $r_2 = 6$, $r_3 = 5$, $r_4 = 2$, $r_5 = 2$. In this case the variables are $b_1, b_2, b_3, b_4, b_5$, their domains are $\{0, 1\}$ and the constraints are $R_{12}, R_{13}, R_{14}, R_{24}, R_{25}, R_{35}$. The cost network for this problem formulation is identical to its constraint network, since all the cost components are unary. The reader can verify that an optimal solution is given by $b_1 = 0, b_2 = 1, b_3 = 1, b_4 = 0, b_5 = 0$. Namely, selecting bids $b_2$ and $b_3$ is an optimal choice with total cost of 11.
Elim-opt for auction problem

processing bucket \(b_4\). In this bucket we compute \(h^4(b_1, b_2) = \max_{(b_4 | (b_1, b_2, b_4) \in R_{41} \times R_{42})} r(b_4)\)

yielding:

\[
h^4(b_1, b_2) = \begin{cases} 
0 & \text{if } b_1 = 1, \text{ or } b_2 = 1 \\
2 & \text{if } b_1 = 0, b_2 = 0 
\end{cases}
\]

Processing bucket \(b_3\) we compute \(h^3(b_1, b_5) = \max_{(b_3 | (b_1, b_3, b_5) \in R_{31} \times R_{35})} r(b_3)\)

yielding:

\[
h^3(b_1, b_5) = \begin{cases} 
0 & \text{if } b_1 = 1, \text{ or } b_5 = 1 \\
5 & \text{if } b_1 = 0, b_5 = 0 
\end{cases}
\]

Processing bucket \(b_2\), which now includes a new function, gives us:

\(h^2(b_1, b_6) = \max_{(b_2 | (b_1, b_2, b_6) \in R_{21} \times R_{23})} (r(b_2) + h^4(b_1, b_2))\)

yielding:

\[
h^2(b_1, b_6) = \begin{cases} 
0 & \text{if } b_1 = 1, b_6 = 1 \\
0 & \text{if } b_1 = 1, b_6 = 0 \\
2 & \text{if } b_1 = 0, b_6 = 1 \\
6 & \text{if } b_1 = 0, b_6 = 0 
\end{cases}
\]

\(b_1 = \{1, 2, 3, 4\}; \ b_2 = \{2, 3, 6\}; \ b_3 = \{1, 5, 4\}; \ b_4 = \{2, 8\}; \ b_5 = \{5, 6\}\)

costs \(r_1 = 8; \ r_2 = 6; \ r_3 = 5; \ r_4 = 2; \ r_5 = 2\)
Bucket-elimination for combinatorial auction

Figure 13.8: Schematic execution of elim-opt on the auction problem
Treating constraints as constraints

Algorithm elim-opt-cons
Input: A cost network $\mathcal{C} = (X, D, C_h, C_s)$, $C_h = \{R_{S_1}, \ldots, R_{S_m}\}$, $C_s = \{F_{Q_1}, \ldots, F_{Q_l}\}$, ordering $d$;
Output: A consistent solution that maximizes $\sum_{F_i \in C_s} F_i$.
1. Initialize: Partition the $C_s$ and $C_h$ into buckets using the usual rule.

2. Process buckets from $p \leftarrow n$ downto $1$,
For costs $h_1, h_2, \ldots, h_j$ defined over scopes $Q_1, \ldots, Q_j$, for hard constraint relations $R_1, R_2, \ldots, R_t$ defined over scopes $S_1, \ldots, S_t$ in bucket $p$, do:

- If (observed variable) $x_p = a_p$, assign $x_p = a_p$ to each $h_i$ and each $R_t$ and put in appropriate buckets.
- Else, (sum and maximize, join and project)
  1. Let $U_p = \cup_i S_i - \{x_p\}$, $V_p = \cup_i Q_i - \{x_p\}$, $W_p = U_p \cup V_p$
  2. $R^p = \pi_{U_p}(\bigwedge_{i=1}^{t} R_i)$. (generate the hard constraint)

3. For every tuple $t$ over $W_p$ do: (generate the cost function)
   $h^p(t) = \max_{(a_p)|(t,a_p)\text{ satisfies } \{R_1,\ldots,R_t\}} \sum_{i=1}^{j} h_i(t, a_p)$.
   Place $h^p$ in the latest lower bucket mentioning a variable in $W_p$. Place $R^p$ in the bucket of the latest variable in $U_p$.

3. Forward: Assign maximizing values in ordering $d$, consulting functions in each bucket.

Figure 13.6: Algorithm elim-opt-cons
Bucket-elimination for counting

Algorithm elim-count
Input: A constraint network $\mathcal{R} = (X, D, C)$, ordering $d$.
Output: Augmented output buckets including the intermediate count functions and The number of solutions.
1. **Initialize**: Partition $C$ (0-1 cost functions) into ordered buckets $bucket_1, \ldots, bucket_n$, We denote a function in a bucket $N_i$, and its scope $S_i$.
2. **Backward**: For $p \leftarrow n$ downto 1, do
   - Generate the function $N^p$: $N^p = \sum X_p \prod N_{i\in bucket_p} N_i$.
   - Add $N^p$ to the bucket of the latest variable in $\bigcup_{i=1}^{p} S_i - \{X_p\}$.
3. **Return** the number of solutions, $N^1$ and the set of output buckets with the original and computed functions.

Figure 13.9: Algorithm elim-count
Outline

- Introduction
  - Optimization tasks for graphical models
  - Solving by inference and search
- Inference
  - Bucket elimination, dynamic programming, tree-clustering, bucket-elimination
  - Mini-bucket elimination, belief propagation
- Search
  - Branch and bound and best-first
  - Lower-bounding heuristics
  - AND/OR search spaces
- Hybrids of search and inference
  - Cutset decomposition
  - Super-bucket scheme
Directional i-consistency
Mini-Bucket Elimination (MAP/MPE)

Split a bucket into mini-buckets $\rightarrow$ bound complexity

\[
\lambda_X(\cdot) = \max_x \prod_{i=1}^{n} f_i(x, \ldots)
\]

\[
\lambda_{X,1}(\cdot) = \max_x \prod_{i=1}^{r} f_i(x, \ldots)
\]

\[
\lambda_{X,2}(\cdot) = \max_x \prod_{i=r+1}^{n} f_i(x, \ldots)
\]

\[
\lambda_X(\cdot) \leq \lambda_{X,1}(\cdot) \lambda_{X,2}(\cdot)
\]

Exponential complexity decrease: $O(e^n) \rightarrow O(e^r) + O(e^{n-r})$
Mini-Bucket Elimination (MAP)

\[ \lambda_{B\rightarrow C}(a, c) = \max_b f(a, b) \cdot f(b, c) \]
\[ \lambda_{B\rightarrow D}(d, e) = \max_b f(b, d) \cdot f(b, e) \]
\[ \lambda_{C\rightarrow E}(a, e) = \max_c \ldots \]

**U = upper bound**

In our case, \( \min \sum \)
Mini-Bucket Decoding (MAP)

\[ b^* = \arg \max_b f(a^*, b) \cdot f(b, c^*) \cdot f(b, d^*) \cdot f(b, e^*) \]

\[ c^* = \arg \max_c f(c, a^*) \cdot f(c, e^*) \cdot \lambda_{B \rightarrow C}(a^*, c) \]

\[ d^* = \arg \max_d f(a^*, d) \cdot \lambda_{B \rightarrow D}(d, e^*) \]

\[ e^* = \arg \max_e \lambda_{C \rightarrow E}(a^*, e) \cdot \lambda_{D \rightarrow E}(a^*, e) \]

\[ a^* = \arg \max_a f(a) \cdot \lambda_{E \rightarrow A}(a) \]

Greedy configuration = lower bound

\( \text{U} = \text{upper bound} \)
Properties of Mini-Bucket Elimination

Bounding from above and below:

- **Complexity**: $O(r \exp(i))$ time and $O(\exp(i))$ space.
- **Accuracy**: determined by Upper/Lower bound.
- As $i$ increases, both accuracy and complexity increase.
- Possible use of mini-bucket approximations:
  - As anytime algorithms
  - As heuristics in search
MBE vs BE

Figure 13.1: The cost graph of the cost function: $C(a, b, c, d, f, g) = F_0(a) + F_1(a, b) + F_2(a, c) + F_3(b, c, f) + F_4(a, b, d) + F_5(f, g)$

(a) A trace of \textit{elim-opt}

(b) A trace of \textit{mbbe-opt}(3)
Bucket-elimination for combinatorial auction example

Figure 13.8: Schematic execution of elim-opt on the auction problem
Example 13.4.3 Let us apply algorithm mbe-opt(2) to the auction problem along the ordering $d = b_1, b_5, b_2, b_3, b_4$. Figure 13.12 shows the resulting mini-buckets; square brackets denote the choice for partitioning.

We start with processing bucket $b_1$. A possible partitioning places the constraint $R_{41}$ in one mini-bucket and the rest in the other. We compute a constraint $R_1(b_1) = \pi_{b_1} R_{41}$, which is the universal constraint so it need not be recorded. In the second mini-bucket, we also compute $h^4(b_2) = \max_{\{b_4 \mid (b_4, b_2) \in R_{42}\}} r(b_1)$, yielding:

$$h^4(b_2) = \begin{cases} 
0 & \text{if } b_2 = 1 \\
2 & \text{if } b_2 = 0 
\end{cases}$$

Processing the first mini-bucket of $b_2$, which includes only hard constraints, will also not effect the domain of $b_1$. Processing the second mini-bucket of $b_2$ by $h^3(b_5) = \max_{\{b_5 \mid (b_5, b_3) \in R_{25}\}} r(b_3)$ yields:

$$h^3(b_5) = \begin{cases} 
0 & \text{if } b_5 = 1 \\
5 & \text{if } b_5 = 0 
\end{cases}$$

Processing the second mini-bucket of $b_3$ (the first mini-bucket includes a constraint whose projection is a universal constraint) by $h^2(b_5) = \max_{\{b_2 \mid (b_2, b_5) \in R_{25}\}} (r(b_2) + h^4(b_2))$, gives us:

$$h^2(b_5) = \begin{cases} 
2 & \text{if } b_5 = 1 \\
6 & \text{if } b_5 = 0 
\end{cases}$$

Processing bucket $b_3$ (with full buckets now) the algorithm computes $h^5 = \max_{b_3}(r(b_5) + h^2(b_5) + h^3(b_5))$, yielding:

$$h^5 = 11$$

Finally, in the bucket of $b_1$ the algorithm computes $h^1 = \max_{b_1}(r(b_1) + h^5)$, yielding:

$$h^1 = \max\{0, 11, 8, 11\} = 19.$$ 

The maximal upper-bound cost is therefore 19. To compute a maximizing tuple we select in bucket $b_1$ the value $b_1 = 1$, which maximizes $r(b_1) + h^5$. Given $b_1 = 1$, we choose $b_5 = 0$, which maximizes the functions in bucket $b_5$. Then, in bucket $b_2$, we can choose only $b_2 = 0$, due to the constraint $R_{12}$. Likewise, the only subsequent choices are $b_3 = 0$ and $b_4 = 0$. Therefore, the cost of the generated solution is 8, yielding the interval $[8, 19]$ bounding the maximal solution. 

\[ \square \]
Algorithm mbe-opt(i)
Input: A cost network $\mathcal{C} = (X, D, C)$; an ordering $d$; parameter i.
Output: An upper bound on the optimal cost solution, a solution and a lower bound and the ordered augmented buckets.
1. **Initialize:** Partition the functions in $\mathcal{C}$ into $\text{bucket}_1, \ldots, \text{bucket}_n$, where $\text{bucket}_i$ contains all functions whose highest variable is $x_i$. Let $S_1, \ldots, S_j$ be the scopes of functions (new or old) in the processed bucket.
2. **Backward** For $p \leftarrow n$ down-to 1, do
   - If variable $x_p$ is instantiated ($x_p = a_p$), assign $x_p = a_p$ to each $h_i$ and put each resulting function into its appropriate bucket.
   - Else, for $h_1, h_2, \ldots, h_j$ in $\text{bucket}_p$, generate an $(i)$-partitioning, $Q' = \{Q_1, \ldots, Q_t\}$.
     For each $Q_t \in Q'$ containing $h_{l_1}, \ldots h_{l_t}$ generate function $h^l$, $h^l = \max_{x_p} \sum_{i=1}^{t} h_{l_i}$.
     Add $h^l$ to the bucket of the largest-index variable in $U_l$, $U_l = \bigcup_{i=1}^{j} \text{scope}(h_{l_i}) - \{x_p\}$.
3. **Forward** For $i = 1$ to $n$ do, given $a_1, \ldots, a_{p-1}$ choose a value $a_p$ of $x_p$ that maximizes the sum of all the functions in $x_p$’s bucket.
4. **Return** the ordered set of augmented buckets, an assignment $\bar{a} = (a_1, \ldots, a_n)$, an interval bound (the value computed in $\text{bucket}_1$ and the cost $F(\bar{a})$).

Figure 13.9: Mini-Bucket Elimination Algorithm
Properties of MBE(i)

- **Complexity:** $O(r \exp(i))$ time and $O(\exp(i))$ space.
- Yields an upper-bound and a lower-bound.

- **Accuracy:** determined by upper/lower (U/L) bound.

- As $i$ increases, both accuracy and complexity increase.

- Possible use of mini-bucket approximations:
  - As *anytime algorithms*
  - As *heuristics* in search

- Other tasks: similar mini-bucket approximations for: *belief updating, MAP and MEU* (Dechter and Rish, 1997)
Outline

- **Introduction**
  - Optimization tasks for graphical models
  - Solving optimization problems with inference and search

- **Inference**
  - Bucket elimination, dynamic programming
  - Mini-bucket elimination

- **Search**
  - Branch and bound and best-first
  - Lower-bounding heuristics
  - AND/OR search spaces
Branch and bound

procedure BRANCH-AND-BOUND
Input: A cost network $C = (X, D, C_h, C_k)$, $L$ current upper-bound, An upper-bound function $f$ defined for every partial solution.
Output: Either an optimal (maximal) solution, or notification that the network is inconsistent.

\begin{align*}
i &← 1 \quad \text{(initialize variable counter)} \\
D_i' &← D_i \quad \text{(copy domain)}
\end{align*}

While $1 ≤ i ≤ n$

\begin{align*}
\text{instantiate } x_i &← \text{SELECTVALUE} \\
\text{If } x_i \text{ is null} &\quad \text{(no value was returned)} \\
\text{else} &\quad \text{(backtrack)}
\end{align*}

\begin{align*}
i &← i + 1 \quad \text{(step forward)} \\
D_i' &← D_i
\end{align*}

Endwhile

If $i = 0$

Return “inconsistent”

Else

Compute $C = C(x_1, ..., x_n)$, $U ← max\{C, L\}$
ic $← n - 1$

\end{procedure}

procedure SELECTVALUE

\begin{align*}
\text{If } i = 0 &\text{ return } U \text{ as the solution value and the most recent assignment as solution.} \\
\text{While } D_i' \text{ is not empty} &\text{ select an element } a ∈ D_i' \text{ having max } f(\bar{a}_{i-1}, a) \\
\text{and remove } a \text{ from } D_i' &\text{ if } < x_i, a > \text{ is consistent with } \bar{a}_{i-1} \text{ and } f > L, \text{ then} \\
\text{Return } a &\text{ (else prune } a) \\
\text{Endwhile}
\end{align*}

Return null \quad \text{(no consistent value)}

\end{procedure}

Figure 13.2: The Branch and Bound algorithm.
Search tree for first-choice heuristic

\[ f_{fc}(\overline{a}_i) = \sum_{F_j \in \mathcal{C}} \max_{a_{i+1}, \ldots, a_n} F_j(\overline{a}_n) \]

Example 13.2.1 Consider the auction problem described in Example 13.1.2. Searching for a solution in the order \( d = b_1, b_2, b_3, b_4, b_5 \) yields the search space in Figure 13.3, traversed from left to right. The search space is highly constrained in this case. The evaluation bounding function at each node should overestimate its best extension for a solution. The first-choice bounding function for the root node (the empty assignment) is 23. The first solution encountered (selecting \( < b_1, 1 > \) and \( < b_5, 1 > \)) has a cost of 10, which becomes the current global lower bound. The next solution encountered has the cost of 11 (when \( < b_2, 1 > \) and \( < b_3, 1 > \)), while the rest of the variables are assigned 0). Subsequently, the partial assignment (\( < b_1, 0 >, < b_2, 0 > \)) is explored. Since \( f_{fc}(< b_1, 0 >, < b_2, 0 >) = 9 \), this upper bound is lower than 11, and therefore search can be pruned.
The search space

Objective function:
The search space

Arc-cost is calculated based on cost components.
The value function

Value of node = minimal cost solution below it
An optimal solution

Value of node = minimal cost solution below it
Basic heuristic search schemes

Heuristic function $f(x)$ computes a lower bound on the best extension of $x$ and can be used to guide a heuristic search algorithm. We focus on

1. **Branch and Bound**
   Use heuristic function $f(x^p)$ to prune the depth-first search tree.
   Linear space

2. **Best-First Search**
   Always expand the node with the highest heuristic value $f(x^p)$.
   Needs lots of memory
Classic branch-and-bound

\[
LB(n) = g(n) + h(n)
\]

Upper Bound \( UB \)
Lower Bound \( LB \)

Prune if \( LB(n) \geq UB \)
How to Generate Heuristics

• The principle of relaxed models
  • Linear optimization for integer programs
  • Mini-bucket elimination
  • Bounded directional consistency ideas
Generating Heuristics for Graphical Models

Given a cost function:
\[ f(a, \ldots, e) = f(a) + f(a, b) + f(a, c) + f(a, d) + f(b, c) + f(b, d) + f(b, e) + f(c, e) \]

define an evaluation function over a partial assignment as the cost of its best extension:

\[ f^*(\hat{a}, \hat{e}, D) = \min_{b,c} F(\hat{a}, b, c, D, \hat{e}) \]

\[ = f(\hat{a}) + \min_{b,c} f(\hat{a}, b) + f(\hat{a}, c) + \cdots \]

\[ = g(\hat{a}, \hat{e}, D) + h^*(\hat{a}, \hat{e}, D) \]

[Kask and Dechter, 2001]
Static Mini-Bucket Heuristics

Given a partial assignment, \( [\hat{a} = 1, \hat{e} = 0] \)
mini-bucket gives an admissible heuristic:

\[
\tilde{h}(\hat{a}, \hat{e}, D) = \lambda_{C \rightarrow E}(\hat{a}, \hat{e}) + f(\hat{a}, D) + \lambda_{B \rightarrow D}(D, \hat{e})
\]
(admissible: \( \tilde{h}(\hat{a}, \hat{e}, D) \leq h^*(\hat{a}, \hat{e}, D) \))

cost to go:
\[
\tilde{h}(\hat{a}, \hat{e}, D) = \lambda_{C \rightarrow E}(\hat{a}, \hat{e}) + f(\hat{a}, D) + \lambda_{B \rightarrow D}(D, \hat{e})
\]
(admissible: \( \tilde{h}(\hat{a}, \hat{e}, D) \leq h^*(\hat{a}, \hat{e}, D) \))

cost so far:
\[
g(\hat{a}, \hat{e}, D) = f(A = \hat{a})
\]

\( L = \text{lower bound} \)
BBMB(i) with mini-bucket heuristics

Let us now apply BnB with $f_{mb}$, which is the bounding function extracted from mb-opt(2). Based on the functions in the augmented bucket of $b_1$ produced by mbe-opt(2), $f_{mb}(b_1 = 0) = r(b_1) + h^5 = 8 + 11 = 19$ while $f_{mb}(b_1 = 1) = 0 + 11 = 11$. Consequently, $b_1 = 1$ is chosen. We next evaluate $f_{mb}(b_1 = 1, b_5) = 8 + h^3(b_5) + h^2(b_5)$, yielding $f_{mb}(b_5 = 0) = 19$ and $f_{mb}(b_5 = 1) = 10$. Consequently, $b_5 = 0$ is chosen. The path is now deterministic, allowing the only choices: $b_2 = b_3 = b_4 = 0$. We end up with a solution having a cost of 8, which becomes the first global lower bound $L = 8$. BnB backtracks. The path is deterministic, dictating the choices ($b_2 = b_3 = b_4 = 0$) whose bounding cost equals 10, yielding a solution with cost 10 ($b_1 = 1, b_5 = 1, b_2 = 0, b_3 = 0, b_4 = 0$). The global lower bound $L$ is updated to 10. The algorithm backtracks to the next choice point, which is ($b_1 = 0$), whose bounding cost is 11. Next, for $b_5$, $f_{mb}(b_1 = 0, b_5) = 0 + r(b_5) + h^2(b_5) + h^3(b_5)$, yielding $f_{mb}(b_5 = 1) = 4$, which can be immediately pruned (less than 10), and $f_{mb}(b_5 = 1) = 11$. We select $b_5 = 0$. Subsequently, choosing a value for $b_2$ we get: $f_{mb}(b_1 = 0, b_5 = 0, b_2) = r(b_2) + h^4(b_2) + h^3(b_5)$ (note that $h^2(b_5)$ is not included since it is created in bucket $b_2$). This yields $f_{mb}(b_2 = 1) = 11$, while $f_{mb}(b_2 = 0) = 5$, which is pruned. The next choice for $b_3$ is determined using the bounding function $f_{mb}(b_1 = 0, b_5 = 0, b_2 = 1, b_3) = 6 + r(b_3) + h^4(b_2 = 1)$, yielding for $b_3 = 1$ the bound 11, while for $b_3 = 0$ the bound 6, which will be pruned. $b_3 = 1$ is selected. Subsequently, only $b_4 = 0$ is feasible and we get the best cost solution of value 11. The global lower bound, $L$ is updated to 11 and BnB will lead to only pruned choices.

We see in this example that the performance of BnB using these two bounding func-
BnB with first-cut (b) and mini-bucket (a) heuristics (BBMB(i))
Heuristics properties

• MBE Heuristic is monotone, admissible
• Retrieved in linear time

• IMPORTANT:
  • Heuristic strength can vary by MBE(i).
  • Higher i-bound $\Rightarrow$ more pre-processing $\Rightarrow$ stronger heuristic $\Rightarrow$ less search.

• Allows controlled trade-off between preprocessing and search
Experimental methodology

**Algorithms**
- BBMB(i) – Branch and Bound with MB(i)
- BBFB(i) - Best-First with MB(i)
- MBE(i)

**Test networks:**
- Random Coding (Bayesian)
- CPCS (Bayesian)
- Random (CSP)

**Measures of performance**
- Compare accuracy given a fixed amount of time - how close is the cost found to the optimal solution
- Compare trade-off performance as a function of time
Empirical evaluation of mini-bucket heuristics, Bayesian networks, coding

Random Coding, $K=100$, noise=0.28

Random Coding, $K=100$, noise=0.32
Max-CSP experiments

(Kask and Dechter, 2000)

<table>
<thead>
<tr>
<th>T</th>
<th>MBE BBMB</th>
<th>MBE BBMB</th>
<th>MBE BBMB</th>
<th>MBE BBMB</th>
<th>MBE BBMB</th>
<th>MBE BBMB</th>
<th>MBE BFMB</th>
<th>MBE BFMB</th>
<th>MBE BFMB</th>
<th>PFC-MRDAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=2</td>
<td>70/0.03</td>
<td>90/0.06</td>
<td>100/0.32</td>
<td>100/2.15</td>
<td>100/15.1</td>
<td>100/116</td>
<td>100/116</td>
<td>100/116</td>
<td>100/116</td>
<td>100/0.08</td>
</tr>
<tr>
<td>i=4</td>
<td>90/12.5</td>
<td>100/0.07</td>
<td>100/0.33</td>
<td>100/2.16</td>
<td>100/15.1</td>
<td>100/116</td>
<td>100/116</td>
<td>100/116</td>
<td>100/116</td>
<td>100/0.07</td>
</tr>
<tr>
<td>i=6</td>
<td>80/0.03</td>
<td>100/0.07</td>
<td>100/0.33</td>
<td>100/2.15</td>
<td>100/15.1</td>
<td>100/116</td>
<td>100/116</td>
<td>100/116</td>
<td>100/116</td>
<td>100/0.03</td>
</tr>
<tr>
<td>i=8</td>
<td>0/-</td>
<td>4/0.35</td>
<td>20/2.28</td>
<td>20/15.6</td>
<td>24/123</td>
<td>100/125</td>
<td>100/130</td>
<td>100/757</td>
<td>100/135</td>
<td>100/757</td>
</tr>
<tr>
<td>i=10</td>
<td>0/-</td>
<td>96/644</td>
<td>92/41</td>
<td>96/69</td>
<td>100/125</td>
<td>100/130</td>
<td>100/130</td>
<td>100/130</td>
<td>100/130</td>
<td>100/130</td>
</tr>
<tr>
<td>i=12</td>
<td>0/-</td>
<td>56/131</td>
<td>88/170</td>
<td>92/135</td>
<td>100/130</td>
<td>100/130</td>
<td>100/130</td>
<td>100/130</td>
<td>100/130</td>
<td>100/130</td>
</tr>
<tr>
<td>i=12</td>
<td>0/-</td>
<td>16/597</td>
<td>60/462</td>
<td>88/344</td>
<td>84/216</td>
<td>100/190</td>
<td>100/2879</td>
<td>100/2879</td>
<td>100/2879</td>
<td>100/2879</td>
</tr>
</tbody>
</table>

Dynamic mbe heuristics

• Rather than pre-compiling, the mini-bucket heuristics can be generated during search
• Dynamic mini-bucket heuristics use the Mini-Bucket algorithm to produce a bound for any node in the search space
  (a partial assignment, along the given variable ordering)
Branch and bound w/ mini-buckets

• **BB with static Mini-Bucket Heuristics (s-BBMB)**
  • Heuristic information is pre-compiled before search. Static variable ordering, prunes current variable

• **BB with dynamic Mini-Bucket Heuristics (d-BBMB)**
  • Heuristic information is assembled during search. Static variable ordering, prunes current variable

• **BB with dynamic Mini-Bucket-Tree Heuristics (BBBT)**
  • Heuristic information is assembled during search. Dynamic variable ordering, prunes all future variables
Empirical evaluation

- Algorithms:
  - Complete
    - BBBT
    - BBMB
  - Incomplete
    - DLM
    - GLS
    - SLS
    - IJGP
    - IBP (coding)

- Measures:
  - Time
  - Accuracy (% exact)
  - #Backtracks
  - Bit Error Rate (coding)

- Benchmarks:
  - Coding networks
  - Bayesian Network Repository
  - Grid networks (N-by-N)
  - Random noisy-OR networks
  - Random networks
# Real World Benchmarks

<table>
<thead>
<tr>
<th>Network</th>
<th># vars</th>
<th>avg. dom.</th>
<th>max dom.</th>
<th>BBBT/BBMB/ IJGP i=2</th>
<th>BBBT/BBMB/ IJGP i=4</th>
<th>BBBT/BBMB/ IJGP i=6</th>
<th>BBBT/BBMB/ IJGP i=8</th>
<th>GLS % [time]</th>
<th>DLM % [time]</th>
<th>SLS % [time]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mildew</td>
<td>35</td>
<td>17</td>
<td>100</td>
<td>100[0.28] 30[10.5] 90[3.59]</td>
<td>100[0.56] 95[0.18] 97[33.3]</td>
<td>- 95[2.32] 95[64.9]</td>
<td>- 100[1.97] 100[1.84]</td>
<td>15 [30.02]</td>
<td>0 [30.02]</td>
<td>90 [30.02]</td>
</tr>
<tr>
<td>Pigs</td>
<td>441</td>
<td>3</td>
<td>3</td>
<td>90[15.2] 0[30.01] 80[0.31]</td>
<td>100[3.73] 60[4.85] 77[0.53]</td>
<td>100[2.36] 80[0.02] 80[1.43]</td>
<td>100[0.58] 95[0.04] 83[6.27]</td>
<td>10 [30.02]</td>
<td>0 [30.02]</td>
<td>0 [30.02]</td>
</tr>
<tr>
<td>CPCS360b</td>
<td>360</td>
<td>2</td>
<td>2</td>
<td>100[0.17] 100[0.04] 100[10.6]</td>
<td>100[0.27] 100[0.03] 100[10.5]</td>
<td>100[0.21] 100[0.03] 100[9.82]</td>
<td>100[0.19] 100[0.03] 100[8.59]</td>
<td>100 [30.02]</td>
<td>100 [30.02]</td>
<td>100 [30.02]</td>
</tr>
</tbody>
</table>

Average Accuracy and Time. 30 samples, 10 observations, 30 seconds
Empirical Results: Max-CSP

- **Random Binary Problems**: \(<N, K, C, T>\)
  - \(N\): number of variables
  - \(K\): domain size
  - \(C\): number of constraints
  - \(T\): Tightness

- **Task**: Max-CSP
BBBT(i) vs BBMB(i), N=100

\[ N = 100, \ K = 5, \ C = 300. \ \omega^* = 33.9. \ 10 \ \text{instances.} \ \text{time} = 600\text{sec.} \]

<table>
<thead>
<tr>
<th>( T )</th>
<th>( i=2 )</th>
<th>( i=3 )</th>
<th>( i=4 )</th>
<th>BBMB</th>
<th>( i=5 )</th>
<th>( i=6 )</th>
<th>( i=7 )</th>
<th>BBBT ( i=2 )</th>
<th>PFC-MPRDAC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># solved time backtracks</td>
<td># solved time backtracks</td>
<td># solved time backtracks</td>
<td># solved time backtracks</td>
<td># solved time backtracks</td>
<td># solved time backtracks</td>
<td># solved time backtracks</td>
<td># solved time backtracks</td>
<td># solved time backtracks</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>150K</td>
<td>150K</td>
<td>150K</td>
<td>115K</td>
<td>115K</td>
<td>8</td>
<td>15</td>
<td>7.73</td>
<td>0.03</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>36</td>
<td>32</td>
<td>24</td>
<td>5.3</td>
<td>38</td>
<td>33</td>
<td>33</td>
<td>14.3</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>980K</td>
<td>880K</td>
<td>650K</td>
<td>130K</td>
<td>870K</td>
<td>434K</td>
<td>114</td>
<td>1.5K</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>

BBBT(i) vs. BBMB(i).
Outline

- **Introduction**
  - Optimization tasks for graphical models
  - Solving optimization problems with inference and search

- **Inference**
  - Bucket elimination, dynamic programming
  - Mini-bucket elimination

- **Search**
  - Branch and bound and best-first
  - Lower-bounding heuristics
  - **AND/OR search spaces**
Classic OR search space

Ordering: A B E C D F
AND/OR search space

Primal graph

DFS tree
AND/OR vs. OR

AND/OR size: exp(4), OR size exp(6)
## OR space vs. AND/OR space

Random graphs with 20 nodes, 20 edges and 2 values per node.

<table>
<thead>
<tr>
<th>width</th>
<th>height</th>
<th>OR space</th>
<th>AND/OR space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time (sec.)</td>
<td>Nodes</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>3.154</td>
<td>2,097,150</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>3.135</td>
<td>2,097,150</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>3.124</td>
<td>2,097,150</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>3.125</td>
<td>2,097,150</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>3.104</td>
<td>2,097,150</td>
</tr>
</tbody>
</table>
AND/OR vs. OR

(A=1, B=1)
(B=0, C=0)
AND/OR vs. OR

(A=1, B=1)
(B=0, C=0)

AND/OR

Space: linear
Time: \(O(\exp(m))\)
\(O(w \cdot \log n)\)

OR

Linear space,
Time:
\(O(\exp(n))\)

Fall 2022
#CSP – AND/OR search tree
#CSP – AND/OR search tree
#CSP – AND/OR Tree DFS

**AND** node: Combination operator (product)

**OR** node: Marginalization operator (summation)
AND/OR Tree search for COP

**AND/OR Tree search for COP**

**Node** = Combination operator (summation)
**OR Node** = Marginalization operator (minimization)
Summary of AND/OR search trees

- Based on a backbone pseudo-tree

- A solution is a subtree

- Each node has a **value** – cost of the optimal solution to the subproblem (computed recursively based on the values of the descendants)

- Solving a task = finding the **value** of the root node

- AND/OR search tree and algorithms are

  ([Freuder & Quinn85], [Collin, Dechter & Katz91], [Bayardo & Miranker95])
  - Space: \( O(n) \)
  - Time: \( O(\exp(m)) \), where \( m \) is the depth of the pseudo-tree
  - Time: \( O(\exp(w \times \log n)) \)
  - BFS is time and space \( O(\exp(w \times \log n)) \)
From search trees to search graphs

• Any two nodes that root identical subtrees or subgraphs can be merged
From search trees to search graphs

- Any two nodes that root **identical** subtrees or subgraphs can be merged
Caching

context(B) = \{A, B\}
context(C) = \{A, B, C\}
context(D) = \{D\}
context(F) = \{F\}
An AND/OR graph: Caching goods
Context-based Caching

- Caching is possible when context is the same

- context = parent-separator set in induced pseudo-graph
  = current variable + parents connected to subtree below

\[
\text{context}(B) = \{A, B\} \\
\text{context}(c) = \{A, B, C\} \\
\text{context}(D) = \{D\} \\
\text{context}(F) = \{F\}
\]
AND/OR Search Graph

Primal graph

Pseudo-tree

class\text{(A)} = \{A\}
class\text{(B)} = \{B,A\}
class\text{(C)} = \{C,B\}
class\text{(D)} = \{D\}
class\text{(E)} = \{E,A\}
class\text{(F)} = \{F\}
AND/OR Search Graph

Primal graph

Pseudo-tree

context(A) = {A}
context(B) = {B, A}
context(C) = {C, B}
context(D) = {D}
context(E) = {E, A}
context(F) = {F}
Context-based caching

context(A) = {A}
context(B) = {B, A}
context(C) = {C, B}
context(D) = {D}
context(E) = {E, A}
context(F) = {F}

Cache Table (C)

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Space: $O(\exp(2))$
AND/OR Search Graph (Optimization)

Objective function: \( F^* = \min_x \sum_{\alpha} f_\alpha(x_\alpha) \)

Context minimal AND/OR search graph
Complexity of AND/OR Graph

- **Theorem:** Traversing the AND/OR search graph is time and space exponential in the *induced width/tree-width*.

- If applied to the OR graph complexity is time and space exponential in the *path-width*. 
Treewidth vs. Pathwidth

Treewidth = \(3\) = (max cluster size) - 1

Pathwidth = \(4\) = (max cluster size) - 1
#CSP – AND/OR search tree

![Search Tree Diagram](image-url)
#CSP – AND/OR tree dfs
#CSP – AND/OR search graph
(Caching goods)
#CSP – AND/OR search graph
(Caching goods)

Time and Space
$O(\exp(w*))$

Space
$O(\exp(sep-w*))$

Fall 2022
All Four Search Spaces

Full OR search tree
126 nodes

Full AND/OR search tree
54 AND nodes

Context minimal OR search graph
28 nodes

Context minimal AND/OR search graph
18 AND nodes
AND/OR vs. OR dfs algorithms

- **AND/OR tree**
  - Space: \(O(n)\)
  - Time: \(O(n k^m)\)
    \(O(n k^{w* \log n})\)

  (Freuder85; Bayardo95; Darwiche01)

- **OR tree**
  - Space: \(O(n)\)
  - Time: \(O(k^n)\)

- **AND/OR graph**
  - Space: \(O(n k^{w*})\)
  - Time: \(O(n k^{w*})\)

- **OR graph**
  - Space: \(O(n k^{pw*})\)
  - Time: \(O(n k^{pw*})\)

\(k = \) domain size
\(m = \) pseudo-tree depth
\(n = \) number of variables
\(w^* = \) induced width
\(pw^* = \) path width
Searching AND/OR graphs

• $AO(i)$: searches depth-first, cache $i$-context
  • $i$ = the max size of a cache table (i.e. number of variables in a context)

\[ i = 0 \quad i = w^* \]

<table>
<thead>
<tr>
<th>Space</th>
<th>Time</th>
<th>( i = 0 )</th>
<th>( i = w^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>$O(n)$</td>
<td></td>
<td>$O(exp \ w^*)$</td>
</tr>
<tr>
<td>Time</td>
<td>$O(exp(w^* \ log n))$</td>
<td></td>
<td>$O(exp \ w^*)$</td>
</tr>
</tbody>
</table>

$AO(i)$ time complexity?
AND/OR branch-and-bound (AOBB)

- Associate each node \( n \) with a static heuristic estimate \( h(n) \) of \( v(n) \)
  - \( h(n) \) is a lower bound on the value \( v(n) \)

- For every node \( n \) in the search tree:
  - \( ub(n) \) – current best solution cost rooted at \( n \)
  - \( lb(n) \) – lower bound on the minimal cost at \( n \)
Lower/Upper bounds

UB(X) = best cost below X (i.e. v(X,0))

LB(X) = LB(X, 1)

LB(X, 1) = l(X, 1) + v(A) + h(C) + LB(B)

LB(B) = LB(B, 0)

LB(B, 0) = h(B, 0)

Prune below AND node (B, 0) if LB(X) ≥ UB(X)
Shallow/deep cutoffs

Prune if \( \text{LB}(X) \geq \text{UB}(X) \)

Reminiscent of Minimax shallow/deep cutoffs
Summary of AOBB

• Traverses the AND/OR search tree in a depth-first manner

• Lower bounds computed based on heuristic estimates of nodes at the frontier of search, as well as the values of nodes already explored

• Prunes the search space as soon as an upper-lower bound violation occurs
Heuristics for AND/OR

• In the AND/OR search space $h(n)$ can be computed using any heuristic. We used:

  • Static Mini-Bucket heuristics

  • Dynamic Mini-Bucket heuristics

  • Maintaining FDAC [Larrosa & Schiex03] (full directional soft arc-consistency)
Empirical evaluation

• Tasks
  • Solving WCSPs
  • Finding the MPE in belief networks

• Benchmarks (WCSP)
  • Random binary WCSPs
  • RLFAP networks (CELAR6)
  • Bayesian Networks Repository

• Algorithms
  • s-AOMB(i), d-AOMB(i), AOMFDAC
  • s-BBMB(i), d-BBMB(i), BBMFDAC
  • Static variable ordering (dfs traversal of the pseudo-tree)
Random binary wcsps
(Marinescu and Dechter, 2005)

Random networks with $n=20$ (number of variables), $d=5$ (domain size), $c=100$ (number of constraints), $t=70\%$ (tightness). Time limit 180 seconds.

AO search is superior to OR search
Random binary wcsp (contd.)

**dense**

n=20 (variables), d=5 (domain size), c=100 (constraints), t=70% (tightness)

(20,5,100,0.7) $w^*=12$, $h=15$

**sparse**

n=50 (variables), d=5 (domain size), c=80 (constraints), t=70% (tightness)

(50,5,80,0.7) $w^*=12$, $h=15$

AOMB for large $i$ is competitive with AOMFDAC
### Resource allocation

Radio Link Frequency Assignment Problem (RLFAP)

<table>
<thead>
<tr>
<th>Instance</th>
<th>BBMFDAC</th>
<th>AOMFDAC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time (sec)</td>
<td>nodes</td>
</tr>
<tr>
<td>CELAR6-SUB0</td>
<td>2.78</td>
<td>1,871</td>
</tr>
<tr>
<td>CELAR6-SUB1</td>
<td>2,420.93</td>
<td>364,986</td>
</tr>
<tr>
<td>CELAR6-SUB2</td>
<td>8,801.12</td>
<td>19,544,182</td>
</tr>
<tr>
<td>CELAR6-SUB3</td>
<td>38,889.20</td>
<td>91,168,896</td>
</tr>
<tr>
<td>CELAR6-SUB4</td>
<td>84,478.40</td>
<td>6,955,039</td>
</tr>
</tbody>
</table>

**CELAR6 sub-instances**

AOMFDAC is superior to ORMFDAC
Bayesian networks repository

<table>
<thead>
<tr>
<th>Network (n,d,w*,h)</th>
<th>Algorithm</th>
<th>i=2</th>
<th>i=3</th>
<th>i=4</th>
<th>i=5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>time</td>
<td>nodes</td>
<td>time</td>
<td>nodes</td>
</tr>
<tr>
<td>Barley (48,67,7,17)</td>
<td>s-AOMB(i)</td>
<td>-</td>
<td>8.5M</td>
<td>-</td>
<td>7.6M</td>
</tr>
<tr>
<td></td>
<td>s-BBMB(i)</td>
<td>-</td>
<td>16M</td>
<td>-</td>
<td>18M</td>
</tr>
<tr>
<td></td>
<td>d-AOMB(i)</td>
<td>-</td>
<td>79K</td>
<td>136.0</td>
<td>23K</td>
</tr>
<tr>
<td></td>
<td>d-BBMB(i)</td>
<td>-</td>
<td>2.2M</td>
<td>-</td>
<td>1M</td>
</tr>
<tr>
<td>Munin1 (189,21,11,24)</td>
<td>s-AOMB(i)</td>
<td>57.36</td>
<td>1.2M</td>
<td>12.08</td>
<td>260K</td>
</tr>
<tr>
<td></td>
<td>s-BBMB(i)</td>
<td>-</td>
<td>8.5M</td>
<td>-</td>
<td>9M</td>
</tr>
<tr>
<td></td>
<td>d-AOMB(i)</td>
<td>66.56</td>
<td>185K</td>
<td>12.47</td>
<td>8.1K</td>
</tr>
<tr>
<td></td>
<td>d-BBMB(i)</td>
<td>-</td>
<td>405K</td>
<td>-</td>
<td>430K</td>
</tr>
<tr>
<td>Munin3 (1044,21,7,25)</td>
<td>s-AOMB(i)</td>
<td>-</td>
<td>5.9M</td>
<td>-</td>
<td>4.9M</td>
</tr>
<tr>
<td></td>
<td>s-BBMB(i)</td>
<td>-</td>
<td>1.4M</td>
<td>-</td>
<td>1.2M</td>
</tr>
<tr>
<td></td>
<td>d-AOMB(i)</td>
<td>-</td>
<td>2.3M</td>
<td>68.64</td>
<td>58K</td>
</tr>
<tr>
<td></td>
<td>d-BBMB(i)</td>
<td>-</td>
<td>33K</td>
<td>-</td>
<td>125K</td>
</tr>
</tbody>
</table>

Time limit 600 seconds

available at http://www.cs.huji.ac.il/labs/compbio/Repository

Static AO is better with accurate heuristic (large i)
Outline

• **Introduction**
  • Optimization tasks for graphical models
  • Solving optimization problems with inference and search

• **Inference**
  • Bucket elimination, dynamic programming
  • Mini-bucket elimination

• **Search**
  • Branch and bound and best-first
  • Lower-bounding heuristics
  • AND/OR search spaces