Improved Parallel Branch-and-Bound solver for Rectangular Maximum Agreement Problem

Ai Kagawa [1] Jonathan Eckstein [2]

[1] Computational Science Initiative, Brookhaven National Laboratory
[2] Department of Management Science and Information Systems and RUTCOR, Rutgers University

1/15/2019

RMA Problem Setting

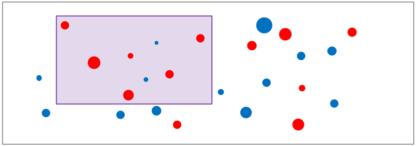
- m observations, each observation has n explanatory variables
- Each observation has a weight $w_i \in \mathbb{R}$
- Explanatory matrix: $X \in \mathbb{R}^{m \times n}$
  - $X_i \in \mathbb{R}^n$ for $i = 1, \ldots, m$ (row vector of X)
  - $x_j \in \mathbb{R}^m$ for $i = 1, \ldots, n$ (column vector of X)
  - $x_{ij} \in \mathbb{R}$: $(i, j)^{th}$ element of X

	x_1	$\cdots$	x_j	$\cdots$	x_n	w
X_1						w_i
$\vdots$						$\vdots$
X_i			x_{ij}			w_i
$\vdots$						$\vdots$
X_m						w_i

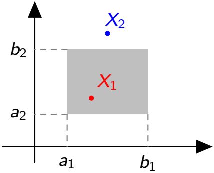
RMA Formulation

- Eliminate zero-weight observations
- Partition observations into

$$\begin{cases} \Omega^+ & = \{i \in \{1, \ldots, m\} \mid w_i > 0\} \\ \Omega^- & = \{i \in \{1, \ldots, m\} \mid w_i < 0\} \end{cases}$$



- Goal: Find an axis-parallel box containing the maximum net weight of positive minus negative observations or vice versa
  - For any set $S \subseteq \{1, \ldots, m\}$, let $w(S) = \sum_{i \in S} w_i$
  - **Coverage** contains indices of observations within (a, b), $\text{Cover}_X(a, b) = \{i \in \{1, \ldots, m\} \mid a \leq X_i \leq b\}$



Rectangular Maximum Agreement (RMA) Problem

$$\begin{array}{ll} \max\limits_{a,b \in \mathbb{R}^n} & \left| w\left(\text{Cover}_X(a, b)\right) \right| \\ \text{s.t.} & a, b \in \mathbb{R}^n \end{array}$$

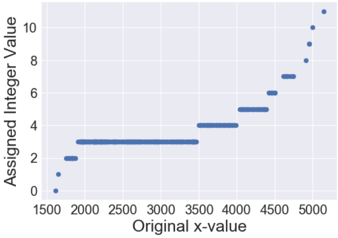
Data Discretization (Preprocessing for RMA)

Link: Branch-and-bound

- Before running RMA, convert $X \in \mathbb{R}^{m \times n} \Rightarrow X \in \mathbb{N}^{m \times n}$
- Assign a natural number to each distinct value of each attribute; aggregation controlled by a parameter $\hat{\delta} = \delta \cdot (\max(x_j) - \min(x_j))$
- $\hat{\delta} = \delta \cdot CI_{.95}(x_j)$ to reduce influence of outliers
- If a range of one assigned value is larger than a certain limit, choose smaller δ for the range and split into smaller clusters (Recursive procedure)
- We discretize uniformly distributed large data into L equal intervals

Non-Recursive Algorithm

$\hat{\delta} = .01 \cdot (\max(x_j) - \min(x_j))$

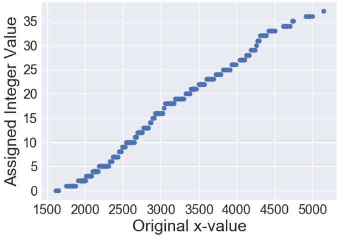


$\Longrightarrow$

Recursive Algorithm

$\hat{\delta} = \delta \cdot CI_{.95}(x_j)$ with $\delta_0 = .1$, $\delta_t = .95\delta_{t-1}$

IntervalLimit $= .05 CI_{.95}(x_j)$



RMA MIP Formulation

Link: RMA Formulation

$$
\begin{aligned}
\max \quad & \phi \\
\text{s.t.} \quad & \phi \le \sum_{i=1}^{m} w_i q_i + \left(\sum_{i=1}^{m} |w_i|\right) 2s, \quad \phi \le -\sum_{i=1}^{m} w_i q_i + \left(\sum_{i=1}^{m} |w_i|\right) 2(1-s) \\
& q_i \le z_{j,x_{ij}} && \forall i, j \\
& q_i \ge \sum_{i=1}^{m} z_{j,x_{ij}} - (n-1), \quad 0 \le q_i \le 1 && \forall i \\
& \sum_{k=0}^{\ell_j - 1} l_{jk} = 1, \quad \sum_{k=0}^{\ell_j - 1} u_{jk} = 1, \quad z_{j,-1} = 0, \quad z_{j,\ell_j} = 0 && \forall j \\
& z_{jk} \le z_{j,k-1} + l_{jk}, \quad z_{jk} \le z_{j,k+1} + u_{jk}, \quad l_{jk} \le z_{jk}, \quad u_{jk} \le z_{jk} && \forall j, k \\
& l_{jk} \in \{0,1\}, \quad u_{jk} \in \{0,1\}, \quad 0 \le z_{jk} \le 1 && \forall j, k
\end{aligned}
$$

ϕ	objective value
s	1 for positive objective value and 0 for negative
q_i	1 if observation i is covered, else 0
u_{jk}	1 if k is the upper bound of attribute j, else 0
l_{jk}	1 if k is the lower bound of attribute j, else 0
z_{jk}	1 if k is a covered value of attribute j, else 0

\# of variables: $m + 2n + 3\sum_{j=1}^{n} \ell_j + 2$

\# of constraints: $mn + m + 4n + 4\sum_{j=1}^{n} \ell_j + 2$

Branch-and-Bound Algorithm

- The Gurobi MIP solver cannot solve large-scale RMA instances Link: MIP
- Maximum Monomial Agreement (MMA) by Eckstein and Goldberg
  - Convert $X \in \mathbb{R}^{m \times n} \Rightarrow X \in \{0, 1\}^{m \times n}$
  - Binarization is too time consuming
- Solve RMA by a specialized parallel branch-and-bound procedure after preprocessing Link: Discretization
- Use PEBBL, a C++ framework for branch-and-bound algorithms (Eckstein, Hart, and Phillips, 2015)

Essential ingredients of all branch-and-bound methods

1. A characterization of subproblems
2. A bounding function for subproblems
3. A branching rule for subdividing subproblems
4. An incumbent (best feasible objective seen so far) computation

- $a \in \mathbb{Z}^n$: lower bound of box
- $b \in \mathbb{Z}^n$: upper bound of box
- Each subproblem P has: $\underline{a}, \overline{a}, \underline{b}, \overline{b} \in \mathbb{Z}^n$

Require:

$$a \leq b$$

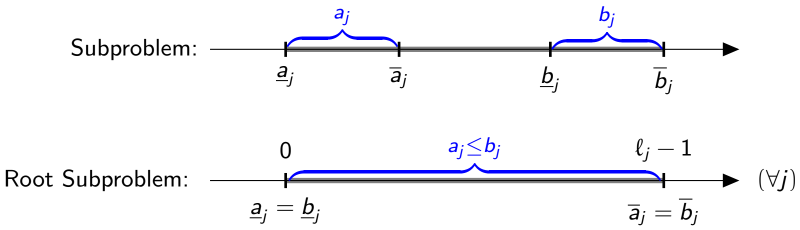
$$\underline{a} \leq a \leq \overline{a} \qquad \underline{a} \leq \overline{a}$$

$$\underline{b} \leq b \leq \overline{b} \qquad \underline{b} \leq \overline{b}$$

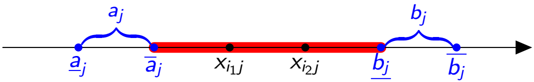
$$\underline{a} \leq \underline{b} \qquad \overline{a} \leq \overline{b}$$

Not Require:

$$\overline{a} \leq \underline{b}$$



Bounding Function and Inseparability



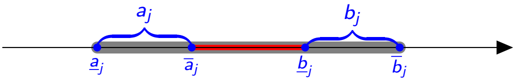
- If either $x_{i_1 j} = x_{i_2 j}$ or $\overline{a}_j \leq x_{i_1 j}, x_{i_2 j} \leq \underline{b}_j$ for each $j = 1, \ldots, n$, then $X_{i_1}, X_{i_2} \in \mathbb{Z}^n$ are **inseparable** w.r.t. $\overline{a}, \underline{b} \in \mathbb{Z}^n$
- $\mathcal{E}(\overline{a}, \underline{b})$: the **equivalence classes** of observation indices induced by the inseparability
- Any box specified by $(\underline{a}, \overline{a}, \underline{b}, \overline{b})$ must either cover or not cover the entirety of each equivalence class $C \in \mathcal{E}(\overline{a}, \underline{b})$

Bounding Function

$$g(\underline{a}, \overline{a}, \underline{b}, \overline{b}) = \max \left\{ \sum_{C \in \mathcal{E}(\overline{a}, \underline{b})} \left[w(C \cap \mathrm{Cover}_X(\underline{a}, \overline{b})) \right]_+ , \sum_{C \in \mathcal{E}(\overline{a}, \underline{b})} \left[w(C \cap \mathrm{Cover}_X(\underline{a}, \overline{b})) \right]_- \right\}$$

- The bound requires computation of the equivalence classes $\mathcal{E}(\overline{a}, \underline{b})$
- Bounds not using these equivalence classes are much weaker

Computing Equivalence Classes



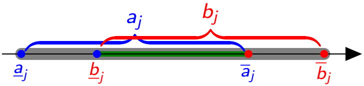
- For a subproblem $P = (\underline{a}, \overline{a}, \underline{b}, \overline{b})$, start with $M = \{1, 2, \ldots, m\}$
- For $j = 1, 2, \ldots, n$, do a stable bucket sort of M on attribute j, but
  - Discard observations with $w_i = 0$
  - Discard observations with $x_{ij} < \underline{a}_j$ or $\overline{b}_j < x_{ij}$
  - Treat $x_{i_1 j}, x_{i_2 j}$ as equal if $\overline{a}_j \leq x_{i_1 j}, x_{i_2 j} \leq \underline{b}_j$
- Time $\mathrm{O}(m)$ per sort, n attributes $\Rightarrow \mathrm{O}(mn)$

for $j = 1 \ldots n$ **do** $M \leftarrow$ **bucketSortObs**$(\underline{a}_j, \overline{a}_j, \underline{b}_j, \overline{b}_j, M, x_j)$ $\mathrm{O}(mn)$

- Scanning through M in order will now yield the equivalence classes: $\mathrm{O}(mn)$ work (but often much less)

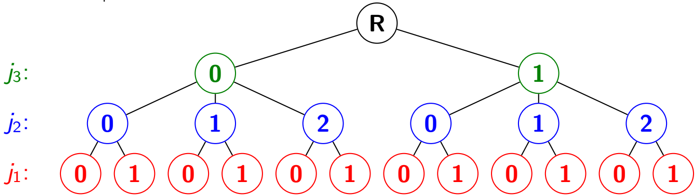
$E \leftarrow$ **createInitEquivClass**$(\underline{a}, \overline{a}, \underline{b}, \overline{b}, M, X)$ $\mathrm{O}(mn)$

First Visual Example: $\underline{b} \leq \overline{a}$



Suppose $\underline{b} \leq \overline{a}$, so that observations are inseparable only if they are identical

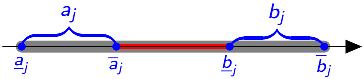
	Observation											
Attribute	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{10}	i_{12}
j_3	0	0	0	0	0	0	1	1	1	1	1	1
j_2	0	0	1	1	2	2	0	0	1	1	2	2
j_1	0	1	0	1	0	1	0	1	0	1	0	1



M :	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9	i_{10}	i_{11}	i_{12}
E :	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}

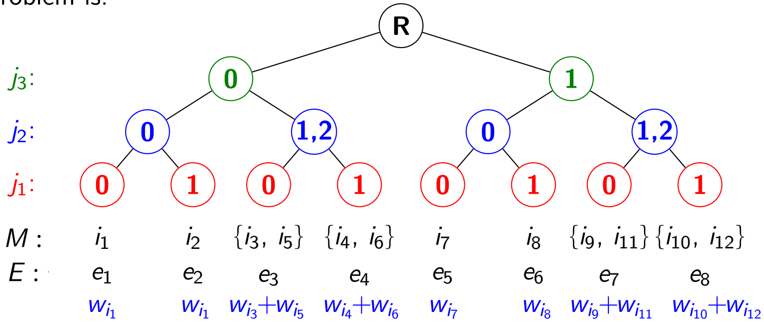
- M: a vector of sorted covered observations
- E: a vector of sorted covered equivalence class indices

Second Visual Example: $\overline{a}_j < \underline{b}_j$



Attribute	$\underline{a}$	$\overline{a}$	$\underline{b}$	$\overline{b}$
j_3	0	1	0	1
j_2	0	**1**	**2**	2
j_1	0	1	0	1

- Values of attribute j_2 are in the same bucket if within $[\overline{a}_{j_2}, \underline{b}_{j_2}] = [1, 2]$
- The initial equivalence class tree for this subproblem is:



- Each equivalence class maintains the sum of observations weights
- At this point, computing the bound is just $O(|E|)$ additional work

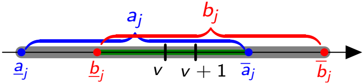
Branching scheme

Link: Branching

Branching of a subproblem $P = (\underline{a}, \overline{a}, \underline{b}, \overline{b})$ is based on:

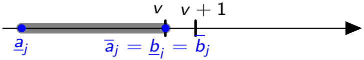
- Pick some feature j and "cut value" $v \in \{\underline{a}_j, \ldots, \overline{b}_j - 1\}$

Parent:

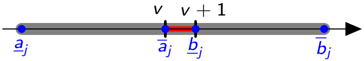


is partitioned into:

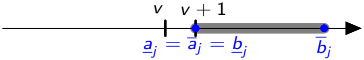
1. **Down Child** — the box is below v in feature j



2. **Middle Child** — the box straddles $[v, v+1]$ in feature j



3. **Up Child** — the box is above $v+1$ in feature j



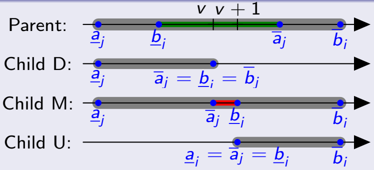
- Except that if $v < \underline{b}_j$, then "down child" is not possible $(b_j \leq v < \underline{b}_j)$
- Else if $\overline{a}_j < v + 1$, then "up child" is not possible $(\overline{a}_j < v + 1 \leq a_j)$
- If $\underline{b}_j < \overline{a}_j$ and $v \in \{\underline{b}_j, \ldots, \overline{a}_j - 1\}$, then 3 children, otherwise 2 children

Child D: Down Child (Box below v)

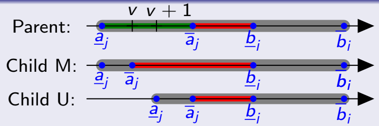
Child M: Middle Child (Box straddles $[v, v+1]$)

Child U: Up Child (Box above $v+1$)

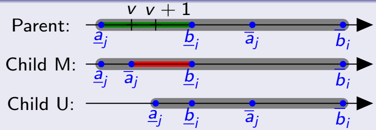
Case: $\underline{b} \leq v \leq \overline{a} - 1$ and $\underline{b} \leq \overline{a}$



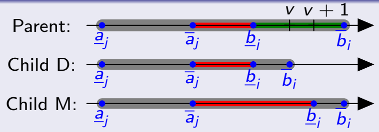
Case: $\underline{a}_j \leq v \leq \overline{a}_j - 1$ and $\overline{a}_j \leq \underline{b}_j$



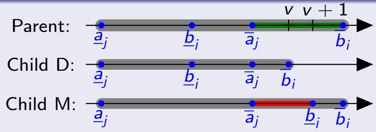
Case: $\underline{a}_j \leq v \leq \underline{b}_j - 1$ and $\underline{b}_j \leq \overline{a}_j$



Case: $\underline{b}_j \leq v \leq \overline{b}_j - 1$ and $\overline{a}_j \leq \underline{b}_j$



Case: $\overline{a}_j \leq v \leq \overline{b}_j - 1$ and $\underline{b}_j \leq \overline{a}_j$



More about Branching

- "**Cutpoint**": each possible attribute / cut value pair (j, v)

How to choose a cutpoint (j, v)

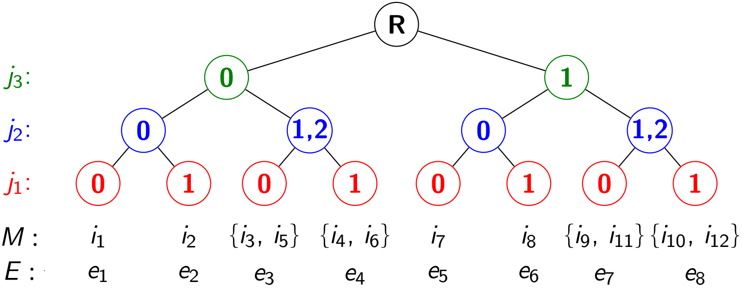
- **Strong branching**: check all possible cutpoints (j, v) and pick the one that $\min_{(j,v)} \{\max\{\text{bounds of 2 or 3 children}\}\}$
- Randomly choose one among cutpoints with tied bounds

- Brute-force implementation (recomputing all the equivalence classes) is $O(mnV) \preceq O(m^2n^2)$ work
  - V is the number of cutpoints $(V \preceq mn)$
  - but only $O(mn^2)$ in the binary-data case
- Instead, exploit relationship between parent and child equivalence classes and use "**rotation algorithm**"

Down Child Bound: Drop Equivalence Classes

- If a child's $\underline{a}_j$ or $\overline{b}_j$ changes, drop equivalence classes no longer covered
- In most recent example, suppose $j = j_1$, $v = 0$
- "Down" child is then: $(\underline{a}_{j_1}, \overline{a}_{j_1}, \underline{b}_{j_1}, \overline{b}_{j_1}) = (0, 0, 0, 0)$

Before Dropping

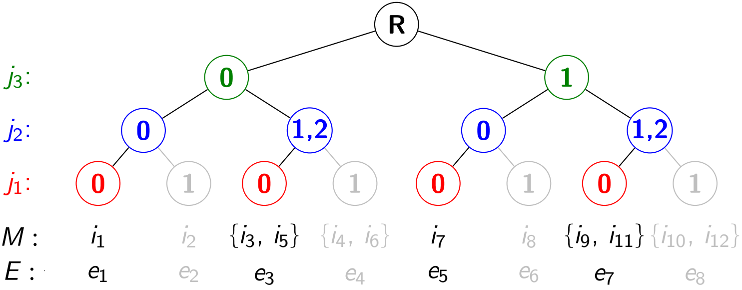


$\widehat{E} \leftarrow \textbf{dropEquivClass}(\underline{a}_j, v, E, x_j) \qquad O(|E|) \preceq O(m)$

Down Child Bound: Drop Equivalence Classes

- If a child's $\underline{a}_j$ or $\overline{b}_j$ changes, drop equivalence classes no longer covered
- In most recent example, suppose $j = j_1$, $v = 0$
- "Down" child is then: $(\underline{a}_{j_1}, \overline{a}_{j_1}, \underline{b}_{j_1}, \overline{b}_{j_1}) = (0, 0, 0, 0)$

After Dropping

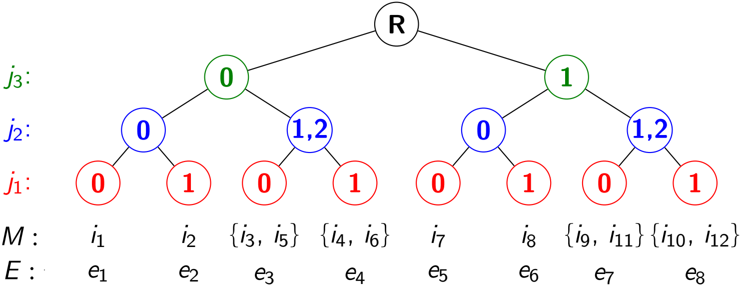


$$\widehat{E} \leftarrow \textbf{dropEquivClass}(\underline{a}_j, v, E, x_j) \qquad O(|E|) \preceq O(m)$$

Up Child Bound: Also Drop Equivalence Classes

- In the same situation, "up" child is $(\underline{a}_{j_1}, \overline{a}_{j_1}, \underline{b}_{j_1}, \overline{b}_{j_1}) = (1,1,1,1)$, so drop different uncovered equivalence classes to obtain

Before Dropping

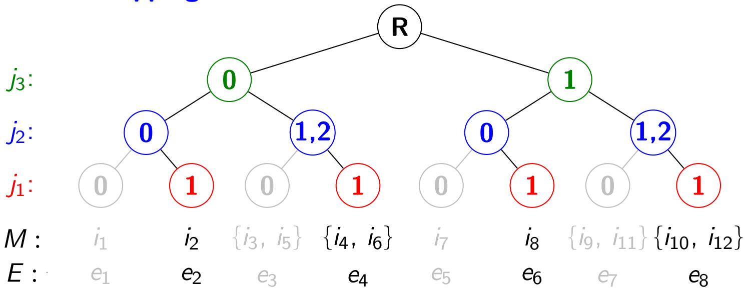


$$\widehat{E} \leftarrow \textbf{dropEquivClass}(v+1, \overline{b}_j, E, x_j) \qquad \mathrm{O}(|E|) \preceq \mathrm{O}(m)$$

Up Child Bound: Also Drop Equivalence Classes

- In the same situation, "up" child is $(\underline{a}_{j_1}, \overline{a}_{j_1}, \underline{b}_{j_1}, \overline{b}_{j_1}) = (1, 1, 1, 1)$, so drop different uncovered equivalence classes to obtain

After Dropping

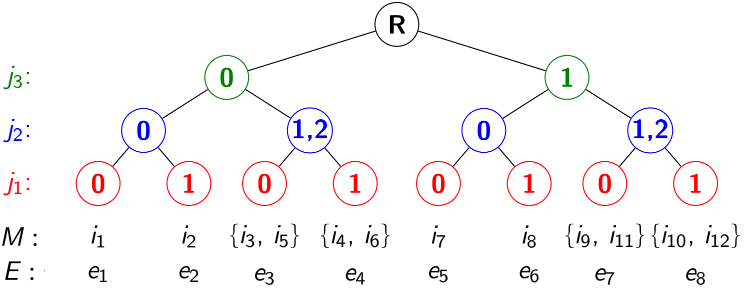


$\widehat{E} \leftarrow$ **dropEquivClass**$(v + 1, \overline{b}_j, E, x_j)$ $\quad O(|E|) \preceq O(m)$

Middle Child: Merge Equivalence Classes

- If a child's $\overline{a}_j$ or $\underline{b}_j$ changes, merge some equivalence classes
- If $j = j_1$, the classes to be merged are adjacent in E
- "Middle" child of our example: $(\underline{a}_{j_1}, \overline{a}_{j_1}, \underline{b}_{j_1}, \overline{b}_{j_1}) = (0, 0, 1, 1)$, and so

Before Merging



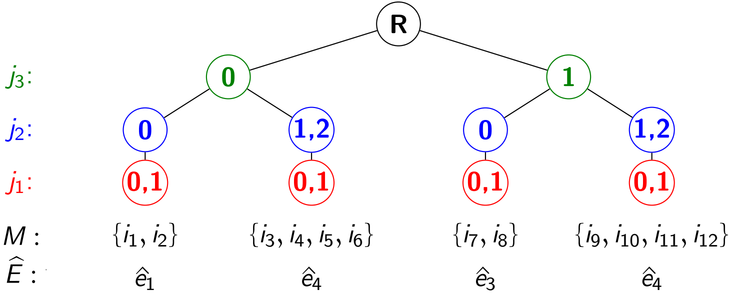
$\widehat{E} \leftarrow$ **mergeEquivClass**(v, j, E, x_j) *

* "Amortized" complexity analysis for this step
Helps to have mergeable classes adjacent

Middle Child: Merge Equivalence Classes

- If a child's $\overline{a}_j$ or $\underline{b}_j$ changes, merge some equivalence classes
- If $j = j_1$, the classes to be merged are adjacent in E
- "Middle" child of our example: $(\underline{a}_{j_1}, \overline{a}_{j_1}, \underline{b}_{j_1}, \overline{b}_{j_1}) = (0, 0, 1, 1)$, and so

After Merging



$\widehat{E} \leftarrow$ **mergeEquivClass**(v, j, E, x_j) *

* "Amortized" complexity analysis for this step
Helps to have mergeable classes adjacent

Evaluate all Potential Children from Cutting on j_1

- For each possible cutpoint (j, v) with $j = j_1$,
  repeat the above procedures to evaluate
  the bounds of the 2 or 3 possible children
- There are at most $\ell_j - 1$ such cutpoints

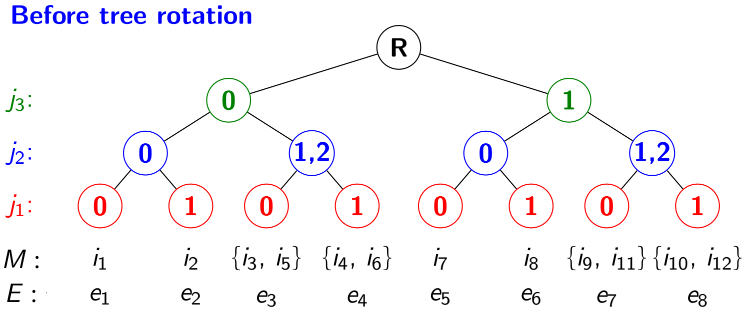
Dropping classes:	$O(\ell_j \lvert E \rvert)$	$\preceq$	$O(m^2)$
Merging classes:	$O(n \lvert E \rvert + \ell_j \lvert E \rvert)$	$\preceq$	$O(nm + m^2)$
Bounds of all children:	$O(\ell_j \lvert E \rvert)$	$\preceq$	$O(m^2)$
Overall:	$O(n \lvert E \rvert + \ell_j \lvert E \rvert)$	$\preceq$	$O(mn + m^2)$

bound $\leftarrow$ **boundComputation** $(j, \underline{a}_j, \overline{a}_j, \underline{b}_j, \overline{b}_j, E, X)$ $O(mn + m^2)$

- To deal with cutpoints with $j \neq j_1 \Rightarrow$ **"rotate" the tree**

Rotate the Tree: Bucket Sort by j_1

- Perform a stable bucket sort of the equivalence classes by attribute j_1



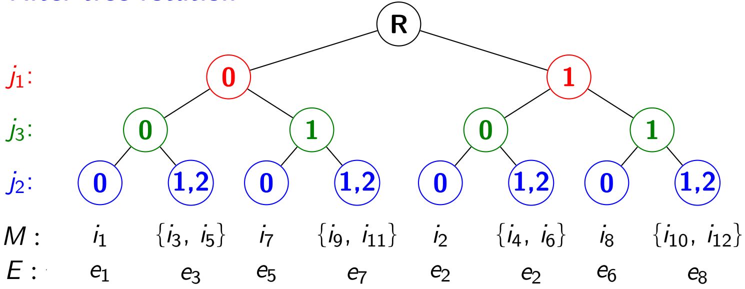
$$E \leftarrow \mathbf{bucketSortE}(\underline{a}_j, \overline{a}_j, \underline{b}_j, \overline{b}_j, E, x_j) \qquad \mathrm{O}(|E|) \preceq \mathrm{O}(m)$$

- Similarly, evaluate all children obtained by cutting on j_2

Rotate the Tree: Bucket Sort by j_1

- Perform a stable bucket sort of the equivalence classes by attribute j_1

After tree rotation



$$E \leftarrow \textbf{bucketSortE}(\underline{a}_j, \overline{a}_j, \underline{b}_j, \overline{b}_j, E, x_j) \qquad O\big(|E|\big) \preceq O(m)$$

- Similarly, evaluate all children obtained by cutting on j_2

Efficiency of Tree Rotation Algorithm

Repeat this procedure for $j = 1, \ldots n - 1$

- Skip for attributes with no possible cutpoints
- Evaluate every possible cutpoint

- Overall Running Time per Subproblem:

$$\mathrm{O}\Big(\underbrace{n}_{n \text{ attributes}} \cdot \underbrace{(mn + m^2)}_{\text{per each attribute}} \Big) = \mathrm{O}(mn^2 + m^2 n)$$

- Usually faster than that: this bound is very pessimistic
  — but still better than $\mathrm{O}(m^2 n^2)$ for brute-force strong branching

Alternatives to Strong Branching

Strong Branching: (checking all possible cutpoints)
— greatly accelerated by the tree rotation technique but still time consuming

Alternative Branching Ideas:

1. **Randomly** choose some percentage of the total cutpoints
   – Did not work well due massive inflation of the branch-and-bound tree
2. **Binary-style search** for a good cutpoint within each feature
   – Only beneficial for attributes with a large number ℓ_j of distinct values
3. **Cutpoint caching:**
   - Many cutpoints are never used
   - Same cutpoints are repeatedly chosen in different parts of the tree

   > Store cutpoints already chosen in earlier subproblems
   > **if** ($\#$ applicable cached cutpoints)
   > $\geq$ (t% of total cutpoins) **then**
   > select the best of them
   > **else** perform strong branching

   - Can combine with binary-style search, depending on ℓ_j

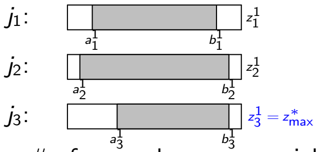
Initial Guess and Greedy Heuristic for RMA

Initially, no constraint $(a, b) = (0, \ell - 1)$, $j^* = -1$, $z^*_{max} = 0$

for each iteration $t = 1, \ldots, T$

1. For each $j \in \{1, \ldots n\} \backslash j^*$, **Kadane's algorithm** finds the range (a^t_j, b^t_j) that optimizes the sum weight z^t_j of covered observations if we narrow the box bounds for attribute j
2. $j^* \leftarrow \arg\max_j \{z^t_j\}$
3. **if** $z^t_{j^*} \leq z^*_{max}$ **then stop** (no improvement)
4. Update $z^*_{max} \leftarrow z^t_{j^*}$ and $(a_{j^*}, b_{j^*}) \leftarrow (a^t_{j^*}, b^t_{j^*})$

Attribute   Iteration 1 (m_1)



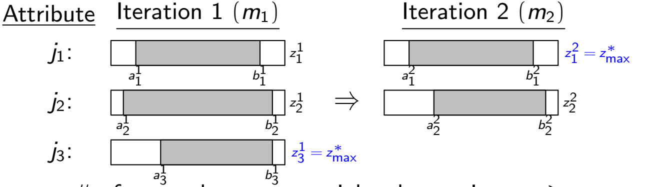
\# of covered non-zero weight observation: $m \geq m_1$

Initial Guess and Greedy Heuristic for RMA

Initially, no constraint $(a, b) = (0, \ell - 1)$, $j^* = -1$, $z^*_{max} = 0$

for each iteration $t = 1, \ldots, T$

1. For each $j \in \{1, \ldots n\} \backslash j^*$, **Kadane's algorithm** finds the range (a^t_j, b^t_j) that optimizes the sum weight z^t_j of covered observations if we narrow the box bounds for attribute j
2. $j^* \leftarrow \arg\max_j \{z^t_j\}$
3. **if** $z^t_{j^*} \leq z^*_{max}$ **then stop** (no improvement)
4. Update $z^*_{max} \leftarrow z^t_{j^*}$ and $(a_{j^*}, b_{j^*}) \leftarrow (a^t_{j^*}, b^t_{j^*})$

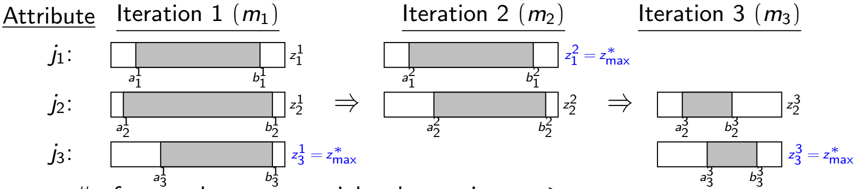


\# of covered non-zero weight observation: $m \geq m_1 > m_2$

Initial Guess and Greedy Heuristic for RMA

Initially, no constraint $(a, b) = (0, \ell - 1)$, $j^* = -1$, $z^*_{max} = 0$
for each iteration $t = 1, \ldots, T$

1. For each $j \in \{1, \ldots n\} \backslash j^*$, **Kadane's algorithm** finds the range (a^t_j, b^t_j) that optimizes the sum weight z^t_j of covered observations if we narrow the box bounds for attribute j
2. $j^* \leftarrow \arg\max_j \{z^t_j\}$
3. **if** $z^t_{j^*} \leq z^*_{max}$ **then stop** (no improvement)
4. Update $z^*_{max} \leftarrow z^t_{j^*}$ and $(a_{j^*}, b_{j^*}) \leftarrow (a^t_{j^*}, b^t_{j^*})$



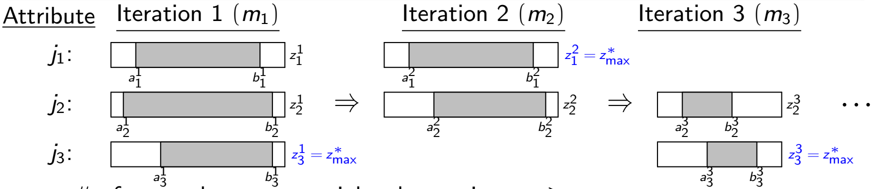
# of covered non-zero weight observation: $m \geq m_1 > m_2 > m_3$

Initial Guess and Greedy Heuristic for RMA

Initially, no constraint $(a, b) = (0, \ell - 1)$, $j^* = -1$, $z^*_{max} = 0$

for each iteration $t = 1, \ldots, T$

1. For each $j \in \{1, \ldots n\} \backslash j^*$, **Kadane's algorithm** finds the range (a^t_j, b^t_j) that optimizes the sum weight z^t_j of covered observations if we narrow the box bounds for attribute j
2. $j^* \leftarrow \arg\max_j \{z^t_j\}$
3. **if** $z^t_{j^*} \leq z^*_{max}$ **then stop** (no improvement)
4. Update $z^*_{max} \leftarrow z^t_{j^*}$ and $(a_{j^*}, b_{j^*}) \leftarrow (a^t_{j^*}, b^t_{j^*})$



\# of covered non-zero weight observation: $m \geq m_1 > m_2 > m_3 > \ldots$

- Repeat this process for the maximum and minimum cases

(incumbent, (a, b)) $\leftarrow$ **initialGuess**(X, y, w) $\quad$ $O\big(mnT\big) \preceq O\big(m^2 n\big)$

Incumbent Heuristic for Each Subproblem

- **Kadane's algorithm** to finds a largest-sum contiguous subvector of any given vector in linear time
- For each attribute j, apply Kadane's algorithm twice (for positive and negative)
- Each Kadane run can be reduced to $\mathrm{O}\big(|E|\big)$

$(\text{incumbent}, (a_j, b_j)) \leftarrow \textbf{checkIncumbent}(j, E, x_j, w) \qquad \mathrm{O}\big(|E|\big) \preceq \mathrm{O}(m)$

- Repeat for each attribute (so $2n$ runs of Kadane in total)
- Total time for checking all attributes $\mathrm{O}\big(n|E|\big) \preceq \mathrm{O}(mn)$

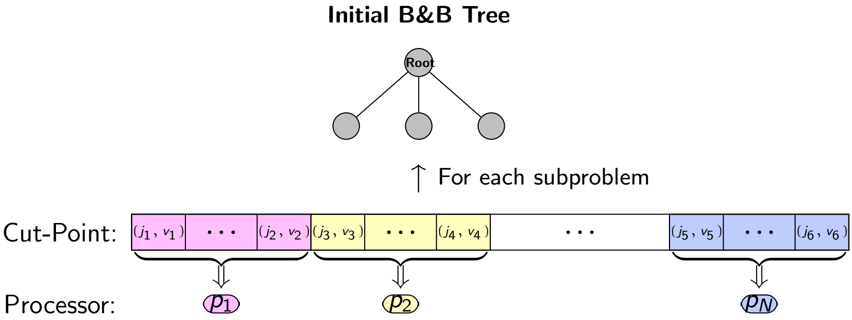
Parallel Implementation: Ramp-up Phase

- Using PEBBL, easy to upgrade the implementation from serial to parallel
- PEBBL's built-in ramp-up / crossover feature

Ramp-up near the tree root,

- Only strong branching and
- All processors cooperate synchronously on evaluating each branch-and-bound node

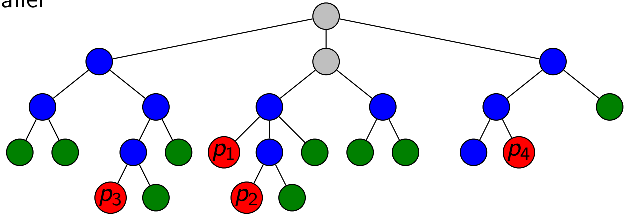
1. Distributes nearly equal number of cutpoints to each processor
2. Each processor computes the bounds for its assigned cutpoints



Parallel Implementation: Crossing Over

Crossover to standard asynchronous search mode:
If (# of active search nodes of B&B tree) $\geq$ (# of processors)
or (# of active search nodes of B&B tree) $>$ (# of cutpoints) **then**

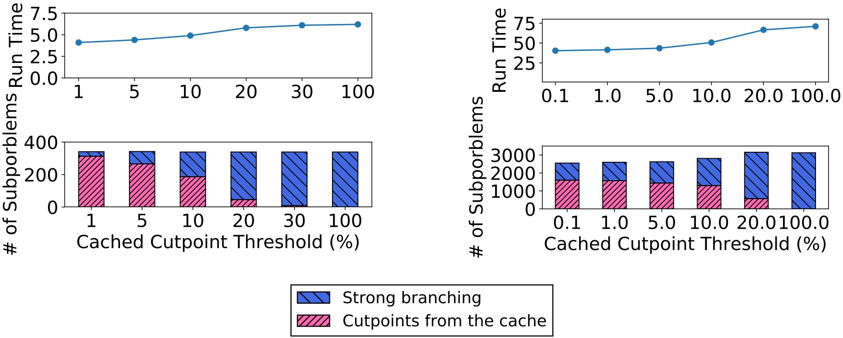
1. PEBBL automatically distributes the current search tree evenly across the available processors
2. Processors start asynchronously evaluating different tree nodes, in parallel



bounded during ramp-up
already bounded by asynchronous search
currently being bounded by asynchrnous search
created but not yet bounded

- Implemented our algorithm in C++ with PEBBL
- Run time improved as we decreased the cutpoint threshold t
- # of total bounded subproblem did not change significantly
- Compute bounds for cutpoitns in cache
  if there is at least 1 applicable cupoint for each subrpoblem

Data: cmc_bin **Data: parkinson**



Results: Benefits of Tree Rotation and Cutpoint Caching

RMA:	RMA with the rotation method and with the strong branching
RMA_CC:	RMA with the rotation method and with the cutpoint caching
RMA_BF:	RMA with brute-force strong branching
MMA:	Binary case of RMA (old implementation by Goldberg)
MIP:	The MIP formulation solved by Gurobi

Serial Run time in seconds for BINARY data sets

Method	cleveland	diabetes	hungheart	cmc_bin	spam	spam75
RMA_CC	**0.5**	**1.0**	**2.0**	**4.1**	**5.2**	**130.2**
RMA	0.5	1.1	3.5	6.1	5.9	154.8
MIP	8.5	14.8	11.5	199.8	276.6	829.4
MMA	20.5	43.6	147.3	909.6	554.4	18966.0
RMA_BF	11.2	70.5	310.0	1104.8	2053.9	☹

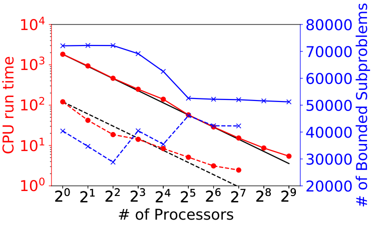
Serial and Parallel Run time in seconds for INTEGER data sets

Method	P	cmc_int	climate	indian	parkinson	skin (245K×3)
RMA_CC	1	**0.7**	21.8	38.4	40.4	**65.9**
RMA	1	1.0	25.3	315.2	68.1	1978.8
MIP	1	64.1	**14.4**	**16.4**	**5.9**	☹
RMA_CC	16	**0.2**	17.9	**4.2**	25.1	**26.7**
RMA	16	**0.2**	**4.1**	26.0	9.5	221.4
MIP	16	12.1	16.2	15.5	**6.3**	☹

- Gurobi generates cutting plans at the root nodes
- PEBBL parallel speedup is much better than Gurobi
- P: # of processors
- Average over 5 runs

Parallel Speedups

Data: indian $(583 \times 10 : 628)$



$(m \times n : V)$

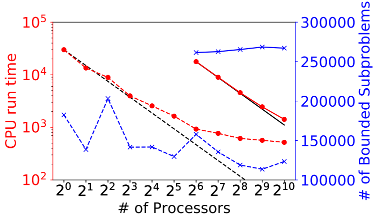
m: # of observations

n: # of attributes

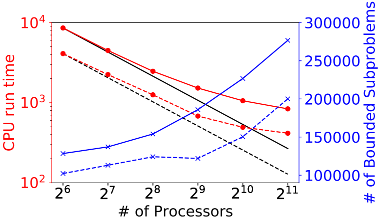
V: # of cutpoints

- RMA Theoretical Speedup
- RMA Run Time
- RMA Subporblems
- RMA_CC Theoretical Speedup
- RMA_CC Run Time
- RMA_CC Subporblems

Data: credit_card $(30K \times 23 : 1931)$



Data: poker $(1M \times 10 : 85)$



- Parallel PEBBL runs nondeterministically
- Average over 5 runs

Conclusion and Planned Improvements/Extensions

Conclusion

- Constructed an improved parallel branch-and-bound procedure and greedy heuristic to solve RMA

Improvement

- Dimensional reduction
- More parallel procedures

Major Contributions

- Developed a **branch-and-bound algorithm** for RMA in C++ programming language with PEBBL, a C++ framework for branch-and-bound algorithms (Eckstein, Hart, and Phillips, 2015)
  - an efficient algorithm to compute an improved bounding function
  - non-strong branching methods
- Built a **greedy heuristic** to solve RMA
- Invented a **recursive data discretization method**
  - Reduces the level of difficulty for RMA