Highly Scalable Parallel Branch and Bound

Jonathan Eckstein (Rutgers)
William Hart (Sandia National Laboratories)
Cynthia A. Phillips (Sandia National Laboratories)
Take Home Message

PEBBL (Parallel Enumeration and Branch and Bound Library) might be a useful tool for solving large-scale problems with a combinatorial piece (e.g. mixed-integer optimization)

Outline

• Branch and Bound
  – Mixed-integer programming
• A combinatorial machine-learning application
  – Scalability study
Branch and Bound

Branch and Bound is an **intelligent** (enumerative) **search** procedure for discrete optimization problems.

\[ \min_{x \in X} f(x) \]

Requires **subproblem representation** and 3 (problem-specific) procedures:

- **Compute an lower bound** \( b(X) \)
  \[ \forall x \in X, \; b(x) \leq f(x) \]

- **Find a candidate solution**
  - Can fail
  - Require that it **recognizes feasibility** if \( X \) has only one point

- **Split** a feasible region (e.g. over parameter/decision space)
  - e.g. Add a constraint
Mixed Integer programming (MIP)

Min $\mathbf{c}^T \mathbf{x}$

Subject to:

$\mathbf{A} \mathbf{x} = \mathbf{b}$

$\ell \leq \mathbf{x} \leq \mathbf{u}$

$x = (x_I, x_C)$

$x_I \in \mathbb{Z}^n$ (integer values)

$x_C \in \mathbb{Q}^n$ (rational values)

- Can also have inequalities in either direction (slack variables):

  $$a_i^T \mathbf{x} \leq b_i \Rightarrow a_i^T \mathbf{x} + s_i = b_i, \ s_i \geq 0$$

- Integer variables represent decisions (1 = yes, 0 = no)
  - Allocation of scarce resources
  - Study of natural systems (mathematics, biology)
Past Applications (Sample)

- Sensor placement (municipal water systems, roadways, buildings)
- Network Interdiction (vulnerability analysis)
- Secure transportation
- Management of unattended ground sensors
  - Volcanoes, subway tunnels, building integrity
- Meshing (for simulating physical systems)
- Space-filling curves - preprocessor for fast heuristic node allocator for MP machines
- Energy system and energy/water planning
- DOE enterprise transformation
- Compliance reviewer allocation
Linear programming (LP)

\[
\begin{align*}
\text{Min} & \quad c^T x \\
\text{Subject to:} & \quad Ax = b \\
& \quad \ell \leq x \leq u \\
& \quad x = (x_I, x_C) \\
& \quad x_I \in \mathbb{Z}^n \quad \text{(integer values)} \\
& \quad x_C \in \mathbb{Q}^n \quad \text{(rational values)}
\end{align*}
\]

- Efficiently solvable in theory and practice
- Gives lower bound for MIP
- LP-based optimization can sometimes give feasible solutions
Branch and Bound

- Recursively divide feasible region, prune search when no optimal solution can be in the region.
- Important: need good bounds, good heuristics
Solution Quality

- Global lower bound (maximum over all active problems): $L = \min_k L_k$
- Approximation ratio for current incumbent $U$ is $U/L$.
- Can stop when $U/L$ is “good enough” (e.g. 105%)
- Running to completion proves optimality
A “good” formulation keeps this region small

- Tight LP
- Added constraints
Branching (Splitting)

- Usually partitions the feasible region (or better)
- Approximately equal difficulty
PEBBL [Eckstein]

Parallel Enumeration and Branch-and-Bound Library
- Distributed memory (MPI), C++

Goals:
- Massively parallel (scalable)
- General parallel Branch & Bound environment
- Parallel search engine cleanly separated from application and platform
- Portable
- Flexible
- Integrate approximation techniques
- Open source

There are other parallel B&B frameworks: PUBB, Bob, PPBB-Lib, Symphony, BCP, CHiPPS/ALPS, FTH-B&B, and codes for MIP
PEBBL Features for Efficient Parallel B&B

- Efficient processor use during ramp-up (beginning)
- Integration of heuristics to generate good solutions early
- Worker/hub hierarchy
- Efficient work storage/distribution
- Control of task granularity
- Load balancing
- Non-preemptive proportional-share “thread” scheduler
- Correct termination
- Early output
- Checkpointing
Pebbl’s Parallelism (Almost) Free

User must

- Define serial application (debug in serial)
- Describe how to pack/unpack data (using a generic packing tool)

C++ inheritance gives parallel management

User may add threads to

- Share global data
- Exploit problem-specific parallelism
- Add parallel heuristics
PEBBL Ramp-up

• Tree starts with one node. What to do with 10,000 processors?
  • Serialize tree growth
    – All processors work in parallel on a single node
  • Parallelize
    – Preprocessing
    – Tough root bounds
    – Incumbent Heuristics
    – Splitting decisions
Pseudocosts

- Compute gradients to help with branching decisions

\[
\begin{align*}
x_j &= 0.3 \\
x_j &= 1
\end{align*}
\]

Down: \[
\frac{15 - 12}{0.3} = 10
\]

Up: \[
\frac{13.4 - 12}{1 - 0.3} = 2
\]

- To initialize, pretend to branch up/down the first time a variable appears fractionally
- Initialize root pseudocosts in parallel

- Branching decision depends on many things: expected bound change (from pseudocosts), user priorities, directional bias, etc.
Classification: Distinguish 2 Classes

- $M$ vectors $v_k$, each with $N$ binary features/attributes: $x_i$ for $i = 1 \ldots N$
- Each vector is a positive or negative example:

$$\Omega^+ \cup \Omega^- = \{1, \ldots, M\} \text{ and } \Omega^+ \cap \Omega^- = \emptyset$$

$$
\begin{array}{cccc|c}
\text{Feature} & x_1 & x_2 & x_3 & x_4 & \text{class} \\
\hline
v_1 & 0 & 0 & 1 & 1 & + \\
\hline
v_2 & 1 & 0 & 0 & 1 & - \\
\hline
v_3 & 1 & 0 & 1 & 1 & - \\
\hline
v_4 & 0 & 1 & 1 & 0 & + \\
\hline
v_5 & 1 & 0 & 0 & 0 & + \\
\end{array}
$$
LP Boosting

- Goal: use labeled examples to infer label of unknowns
- weak learners: $h_j(x) \in \{-1, 0, 1\}$
- Linear combination: $f(x) = \sum_{j=1}^{J} \alpha_j h_j(x)$
  - Sign of $f(x)$ classifies $x$.
  - Use linear programming (LP) to find weights ($\alpha$)
  - Such that $f(x)$ correct on labeled examples
    - Far enough away from zero (really call it)
    - A few mistakes allowed
- Over all possible weak learners.
Separation for Linear Programming

- LP (dual) has an exponential-sized constraint family
- Can still provably enforce the whole family of constraints by explicitly listing only a polynomial number of them
  - Others are redundant
- Needs a separation algorithm:
  - Return a violated cut, or
  - Determine that all cuts are satisfied
Finding most violated weak learner

- $M$ vectors $v_k$, each with $N$ binary features/attributes: $x_i$ for $i = 1 \ldots N$
- Each vector can have a weight $w_i$
- Each vector is a positive or negative example:

$$\Omega^+ \cup \Omega^- = \{1, \ldots, M\} \text{ and } \Omega^+ \cap \Omega^- = \emptyset$$

<table>
<thead>
<tr>
<th>Feature</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>class</th>
<th>$w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+</td>
<td>1.0</td>
</tr>
<tr>
<td>$v_2$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>2.0</td>
</tr>
<tr>
<td>$v_3$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>3.0</td>
</tr>
<tr>
<td>$v_4$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>+</td>
<td>4.0</td>
</tr>
<tr>
<td>$v_5$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+</td>
<td>6.0</td>
</tr>
</tbody>
</table>

Observation Matrix $A$
A binary monomial is a conjunction of binary features: \( x_1 \land \neg x_2 \land x_5 \).

It is equivalent to a binary function:

- Let \( J \) be the set of literals that appear (uncomplemented).
- Let \( C \) be the set of literals that appear complemented.

\[ m_{J,C}(x) = \prod_{j \in J} x_j \prod_{c \in C} (1 - x_c) \]

A binary monomial covers a vector if \( m_{J,C}(x) = 1 \).

The vector agrees with the monomial on each selected feature.

\[ \text{Cover}(J, C) = \left\{ i \in \{1, \ldots, M\} \mid m_{J,C}(A_i) = 1 \right\} \]
Example: Coverage

- Uncomplemented variables $J = \{1\}$ so want $x_1 = 1$
- Complemented variables $C = \{2\}$ so want $x_2 = 0$
- $\text{Cover}(J, C) = \{2, 3, 5\}$ (rows that match criteria)

$x_1 = 1$ and $x_2 = 0$

<table>
<thead>
<tr>
<th>Feature</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>class</th>
<th>$w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+</td>
<td>1.0</td>
</tr>
<tr>
<td>$v_2$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>2.0</td>
</tr>
<tr>
<td>$v_3$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>3.0</td>
</tr>
<tr>
<td>$v_4$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>+</td>
<td>4.0</td>
</tr>
<tr>
<td>$v_5$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+</td>
<td>6.0</td>
</tr>
</tbody>
</table>
**Maximum Monomial Agreement**

- **Maximize** $\text{JC}: \quad g(J, C) = |w(\text{Cover}(J, C) \cap \Omega^+) - w(\text{Cover}(J, C) \cap \Omega^-)|$
  - Weighted difference between covered + and - examples

\[ x_1 = 1 \text{ and } x_2 = 0 \]
\[ g(\{1\}, \{2\}) = 6 - 3 - 2 = 1 \]

**Weak learner:**
- uncovered = 0
- covered = 1

<table>
<thead>
<tr>
<th>Feature</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>class</th>
<th>$w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>+</td>
<td>1.0</td>
</tr>
<tr>
<td>$v_2$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>2.0</td>
</tr>
<tr>
<td>$v_3$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>3.0</td>
</tr>
<tr>
<td>$v_4$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>+</td>
<td>4.0</td>
</tr>
<tr>
<td>$v_5$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+</td>
<td>6.0</td>
</tr>
</tbody>
</table>
B&B Representation for MMA

- Goldberg and Shan (2007) showed best to solve MMA exactly
- Subproblem (partial solution) = (J,C,E,F)
  - J are features forced into monomial
  - C are features forced in as complemented
  - E are eliminated features: cannot appear
  - F are free features
- Any partition of \{1, ..., N\} is possible
- A feasible solution that respects (J,C,E,F) is just (J,C)
- When F is empty, only one element (leaf)

Upper Bound

- Valid: \[ \max \left\{ w(Cover(J, C) \cap \Omega^+), w(Cover(J, C) \cap \Omega^-) \right\} \]

- Strengthen by considering excluded features \( E \)
- Two vectors inseparable if they agree on all features \( i \not\in E \)
  - Creates \( q(E) \) equivalence classes

\[
\begin{array}{cccc}
\text{x}_1 & \text{x}_2 & \text{x}_3 & \text{x}_4 \\
\text{v}_1 & 0 & 0 & 1 & 1 \\
\text{v}_2 & 1 & 0 & 0 & 1 \\
\text{v}_3 & 1 & 0 & 1 & 1 \\
\text{v}_4 & 0 & 1 & 1 & 0 \\
\text{v}_5 & 1 & 0 & 0 & 0 \\
\end{array}
\quad \rightarrow \quad
\begin{array}{cccc}
\text{x}_1 & \text{x}_2 & \text{x}_3 & \text{x}_4 \\
\text{v}_1 & 0 & 1 & 0 & 1 \\
\text{v}_2 & 0 & 1 & 1 & 0 \\
\text{v}_3 & 1 & 0 & 0 & 1 \\
\text{v}_4 & 1 & 0 & 0 & 0 \\
\text{v}_5 & 1 & 1 & 0 & 1 \\
\end{array}
\]
Upper Bound

- \( V^E_\eta \) are vectors in the \( \eta^{th} \) equivalence class
  - All covered or all not covered
    \[
    w^+_\eta(J, C, E) = w(V^E_\eta \cap \text{Cover}(J, C) \cap \Omega^+ )
    \]
    \[
    w^-_\eta(J, C, E) = w(V^E_\eta \cap \text{Cover}(J, C) \cap \Omega^- )
    \]

- Stronger upper bound:
  \[
  b(J, C, E) = \max \left\{ \sum_{\eta=1}^{q(E)} \left( w^+_\eta(J, C, E) - w^-_\eta(J, C, E) \right) \right\}
  \]
Branching

\((J, C, E, F) \text{ and } f \in F\)

\((J \cup \{f\}, C, E, F - \{f\})\)

\((J, C \cup \{f\}, E, F - \{f\})\)

\((J, C, E \cup \{f\}, F - \{f\})\)
Choose branch variable

- Strong branching: for all $f$
  - Compute all 3 upper bounds, $(b_1,b_2,b_3)$ sorted descending
  - Sort lexicographically, pick smallest. Gives lookahead bound

\[
\begin{align*}
  b(J, C, E, F) &= \min_{f \in F} \max \left\{ b(J \cup \{f\}, C, E, F - \{f\}), b(J, C \cup \{f\}, E, F - \{f\}), b(J, C, E \cup \{f\}, F - \{f\}) \right\}
\end{align*}
\]
PEBBL Ramp-up

• Tree starts with one node. What to do with 10,000 processors?
  • Serialize tree growth
    – All processors work in parallel on a single node
  • Parallelize
    – Preprocessing
    – Tough root bounds
    – Incumbent Heuristics
    – Splitting decisions (MMA)
      • Strong-branching for variable selection
PEBBL Ramp-up

- Strong branching for variable selection
  - Divide free variables evenly
  - Processors compute bound triples for their free variables
  - All-reduce on best triples to determine branch var
  - All-reduce to compute lookahead bound

\[
b(J, C, E, F) = \min_{f \in F} \max \left\{ \begin{array}{c}
b(J \cup \{f\}, C, E, F - \{f\}) \\
b(J, C \cup \{f\}, E, F - \{f\}) \\
b(J, C, E \cup \{f\}, F - \{f\})
\end{array} \right\}
\]

- Note: last element most computation: recompute equivalence classes
Experiments

- UC Irvine machine learning repository
  - Hungarian heart disease dataset ($M = 294$, $N = 72$)
  - Spam dataset ($M = 4601$, $N = 75$)
  - Multiple MMA instances based on boost iteration
    - Later iterations are harder
- Dropped observations with missing features
- Binarization of real features (Boros, Hammer, Ibaraki, Kogan)
  - Feature $(i,j)$ is 1 iff $x_i \geq t_j$
  - Cannot map an element of $\Omega^+$ and $\Omega^-$ to the same vector

\[
\begin{align*}
\min & \\
000 & t_1 & 001 & t_2 & 011 & t_3 & 111 & \max
\end{align*}
\]
Red Sky

- Node: two quad-core Intel Xeon X5570 procs, 48GB shared RAM
- Full system: 22,528 cores, 132TB RAM
- General partition: 17,152 cores, 100.5TB RAM
  - Queue wait times OK for 1000s of processors
- Network: Infiniband, 3D torroidal (one dim small), 10GB/s
- Red Hat Linux 5, Intel 11.1 C++ compiler (O2), Open MPI 1.4.3
Value of ramp up (no enumeration)

hung253

Time (Seconds)

Processor Cores

+ + Observations, ramp-up factor 0.0
- - Averages, ramp-up factor 0.0
x x Observations, ramp-up factor 1.0
- - Averages, ramp-up factor 1.0
- - Linear Speedup
Number of tree nodes

hung253

 Processor Cores

Subproblems Bounded

Observations, ramp-up factor 0.0
Averages, ramp-up factor 0.0
Observations, ramp-up factor 1.0
Averages, ramp-up factor 1.0
Spam, value of ramp up

![Graph showing the relationship between time and processor cores](image)
Spam, tree nodes

![Graph showing the relationship between subproblems bounded and processor cores. The graph plots subproblems bounded on the y-axis and processor cores on the x-axis. There are two lines and two sets of data points, each with different ramp-up factors.]

August 31, 2016
SAMSI Optimization Opening Workshop
Comments: Ramp up

- Using initial synchronous ramp up improves scalability (e.g. 2x processors), reduces tree inflation.
- Speed up departure point from linear depends on problem difficulty and tree size.
  - Tree inflation is the main contributor to sub-linear speedup
- Solution times down to 1-3 minutes
  - Spam26: 3 min on 6144 cores, 27 hours on 8 cores
- For MMA no significant efficiency drop from 1 processor and going to multiple hubs
Parallel Enumeration

- Why Enumeration for MMA?
  - MMA is the weak learner for LP-Boost
  - Add multiple violated inequalities
    - In this case, add the best 25 MMA solutions
- Fundamental in PEBBL: best k, absolute tolerance, relative tolerance, objective threshold
- Requires: branch-through on “leaves” and duplicate detection
- Hash solution to find owning processor
- For all but best-k
  - independent solution repositories
  - parallel merge sort at end
- For k-best need to periodically compute cut off objective value
Enumeration Experiments

• Why Enumeration for MMA?
  – MMA is the weak learner for LP-Boost
  – Add multiple columns in column generation
    • In this case, add the best 25 MMA solutions

• Hungarian Heart
  – Tree size about same
  – More communication

• Spam
  – Larger tree with enumeration
  – Harder subproblems than Hungarian heart (more observations)
Results: Enumeration

hung253, enumCount=25

Time (Seconds)

Processor Cores

+ Observations
- Averages
-- Linear Speedup
Results: Enumeration

spam26, enumCount=25
Open-Source Code Available

• Software freely available (BSD license)
  – PEBBL plus knapsack and MMA examples
• http://software.sandia.gov/acro
• ACRO = A Common Repository for Optimizers