Subtraction Strategies: Counting On/Back By Bases and then Ones (CBBO)
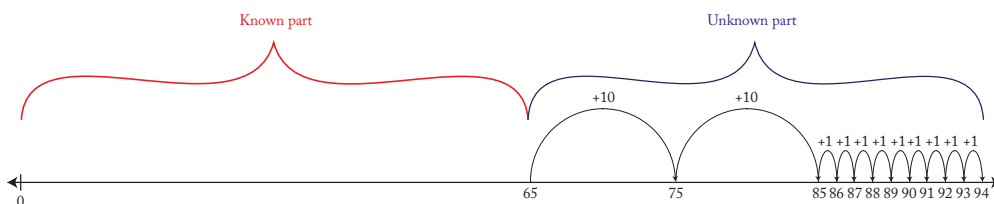
Compiled by: Theodore M. Savich

March 28, 2025
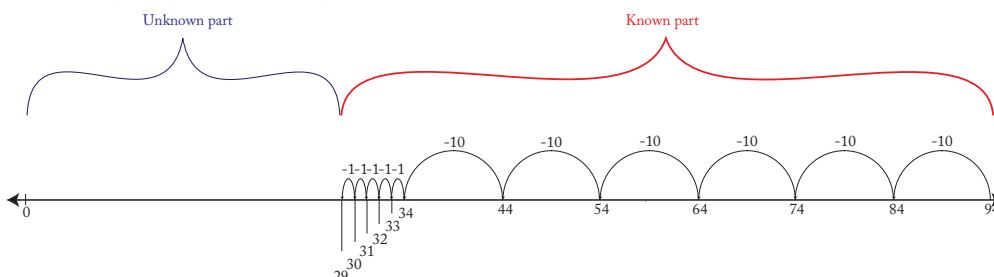
Transcript

Video from Carpenter et al. (1999). Strategy descriptions and examples adapted from Hackenberg (2025).

- **Teacher:** Earl had a collection of 65 bird feathers, on a trip to a marsh he found lots more feathers to put in his collection. Now he has 94 feathers in his collection. How many feathers did Earl find at the marsh?

- **Rita** So he had what?

- **Teacher:** He started off with, 65 feathers.

- **Rita:** 1,2,3,4,5,6 1,2,3,4,5. And then he had how many?

- **Teacher:** Well, he had 65 bird feathers. On a trip to a marsh, he found lots more and he put them in his collection. Now he has 94.

- **Rita:** Well, I can 65, 75, 85. How many did he find?

- **Teacher:** Well, that's my question for you. How many did he find? He ends up with 94.

- **Rita:** And 85,86,87,88,89,90, 91,92,93,94 and so the answer is 20, 21, 22, 23, 24, 25, 26, 27, 28, 29.

- **Teacher** Nice work.

Rita's Way: Counting On by Bases and then Ones (COBO)

Known part   Unknown part

+10   +10

+1 +1 +1 +1 +1 +1 +1 +1 +1

0   65   75   85 86 87 88 89 90 91 92 93 94

Alternatively, Rita could have Counted Back by Bases and Ones (CBBO)

Unknown part   Known part

-10   -10   -10   -10   -10   -10

-1-1-1-1-1

0   34   44   54   64   74   84   94
        33
       32
      31
     30
   29

Notation Representing Rita's Solution:

$$65 + (10) = 75$$
$$75 + (10) = 85$$
$$85 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 94$$
$$10 + 10 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 29$$
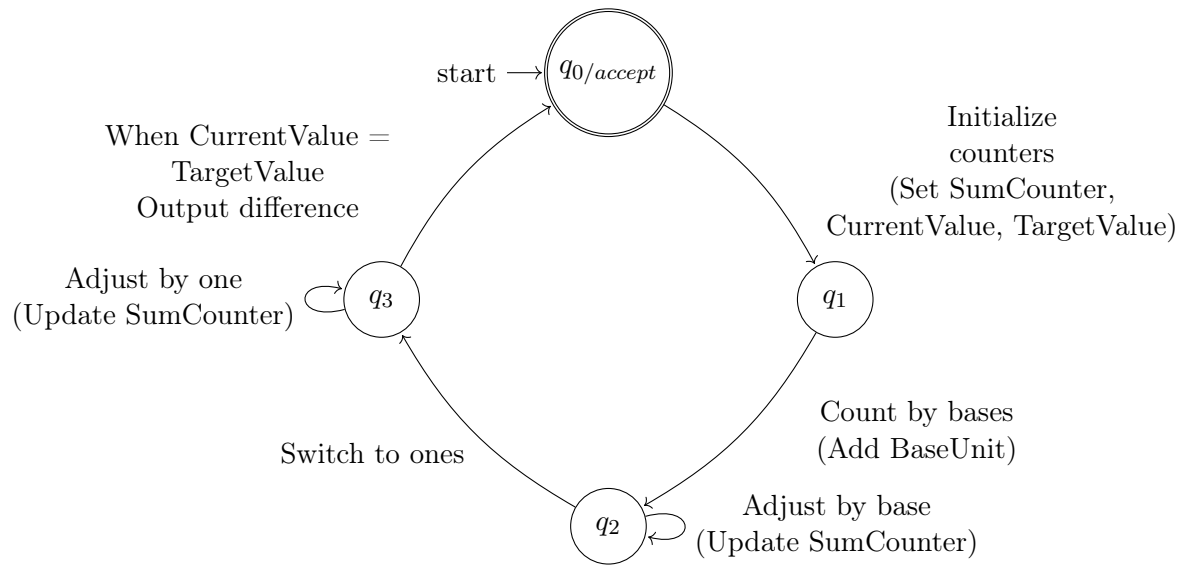
Description of Strategy:

Objective: Description of Counting On by Bases and Then Ones (COBO) Begin with one of the numbers. Break the other number into its base units and its ones. Then, "count on" by adding each base unit one at a time, followed by each individual one.

Why are number lines useful for demonstrating this strategy? COBO is essentially a jump strategy—you start at one number and make "jumps" equal to the other number's base units, then add in the remaining ones. Number lines are ideal because they visually display jumps of varying lengths and directions. They serve as a picture of the process: a jump representing a full base is clearly larger (by a factor of the base) than a jump of a single unit.

Good number line illustrations should:

- Clearly represent the relative sizes of the jumps—each base jump should be exactly as many times larger than a single-unit jump as the base indicates, with all base jumps the same size and all one-unit jumps identical.

- Indicate the position of 0, or mark a break if that portion of the line isn't drawn to scale.

- Use arrows to indicate direction—when adding, the jumps go to the right (or upward); when subtracting, they go to the left (or downward).

- Mark all landing points clearly—the numbers you would speak aloud when counting on by bases and then ones, just as Lauren demonstrated.

Automaton Diagram for Counting On or Back by Bases and Then Ones

start $\longrightarrow$ $q_{0/accept}$

When CurrentValue =
TargetValue
Output difference

Initialize
counters
(Set SumCounter,
CurrentValue, TargetValue)

Adjust by one
(Update SumCounter) q_3

q_1

Switch to ones

Count by bases
(Add BaseUnit)

q_2

Adjust by base
(Update SumCounter)

HTML Implementation

```html
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Subtraction Strategies: Counting Back By Bases and Ones (CBBO)</title>
5      <style>
6          body { font-family: sans-serif; }
7          #diagramCBBOSVG { border: 1px solid #d3d3d3; }
8          #outputContainer { margin-top: 20px; }
9          .number-line-tick { stroke: black; stroke-width: 1; }
10         .number-line-break { stroke: black; stroke-width: 1; } /* Solid for zig-zag */
11         .number-line-label { font-size: 12px; text-anchor: middle; }
12         .jump-arrow { fill: none; stroke: purple; stroke-width: 1.5; } /* CBBO color */
13         .jump-arrow-head { fill: purple; stroke: purple; } /* CBBO color */
14         .jump-label { font-size: 10px; text-anchor: middle; fill: purple; } /* CBBO color
               */
15         .tens-jump-label { font-size: 12px; text-anchor: middle; fill: purple; }
16         .stopping-point { fill: red; stroke: black; stroke-width: 1; }
17         .number-line-arrow { fill: black; stroke: black; }
18         .extended-tick { stroke: black; stroke-width: 1; } /* Reuse COBO style */
19     </style>
20 </head>
21 <body>
22
23 <h1>Subtraction Strategies: Counting Back By Bases and Then Ones (CBBO)</h1>
24
25 <div>
26     <label for="cbboMinuend">Minuend:</label>
27     <input type="number" id="cbboMinuend" value="94"> <!-- Example from PDF -->
28 </div>
29 <div>
30     <label for="cbboSubtrahend">Subtrahend:</label>
31     <input type="number" id="cbboSubtrahend" value="29"> <!-- 94 - 65 = 29 -->
32 </div>
33
34 <button onclick="runCBBOAutomaton()">Calculate and Visualize</button>
35
36 <div id="outputContainer">
37     <h2>Explanation:</h2>
38     <div id="cbboOutput">
39         <!-- Text output will be displayed here -->
40     </div>
41 </div>
42
43 <h2>Diagram:</h2>
44 <svg id="diagramCBBOSVG" width="700" height="350"></svg>
45
46 <script>
47 document.addEventListener('DOMContentLoaded', function() {
48     const outputElement = document.getElementById('cbboOutput');
49     const cbboMinuendInput = document.getElementById('cbboMinuend');
50     const cbboSubtrahendInput = document.getElementById('cbboSubtrahend');
51     const diagramCBBOSVG = document.getElementById('diagramCBBOSVG');
```

```javascript
52
53      // --- Helper Functions (Keep createText, drawTick, drawScaleBreakSymbol,
           createJumpArrow, drawStoppingPoint from previous corrected versions) ---
54      function createText(svg, x, y, textContent, className = 'number-line-label') {
55          const text = document.createElementNS("http://www.w3.org/2000/svg", 'text');
56          text.setAttribute('x', x);
57          text.setAttribute('y', y);
58          text.setAttribute('class', className);
59          text.setAttribute('text-anchor', 'middle'); // Center labels
60          text.textContent = textContent;
61          svg.appendChild(text);
62      }
63
64      function drawTick(svg, x, y, size) {
65          const tick = document.createElementNS('http://www.w3.org/2000/svg', 'line');
66          tick.setAttribute('x1', x);
67          tick.setAttribute('y1', y - size / 2);
68          tick.setAttribute('x2', x);
69          tick.setAttribute('y2', y + size / 2);
70          tick.setAttribute('class', 'number-line-tick');
71          svg.appendChild(tick);
72      }
73
74       function drawScaleBreakSymbol(svg, x, y) {
75          const breakOffset = 4;
76          const breakHeight = 8;
77          const breakLine1 = document.createElementNS('http://www.w3.org/2000/svg', 'line');
78          breakLine1.setAttribute('x1', x - breakOffset);
79          breakLine1.setAttribute('y1', y - breakHeight);
80          breakLine1.setAttribute('x2', x + breakOffset);
81          breakLine1.setAttribute('y2', y + breakHeight);
82          breakLine1.setAttribute('class', 'number-line-break');
83          svg.appendChild(breakLine1);
84          const breakLine2 = document.createElementNS('http://www.w3.org/2000/svg', 'line');
85          breakLine2.setAttribute('x1', x + breakOffset);
86          breakLine2.setAttribute('y1', y - breakHeight);
87          breakLine2.setAttribute('x2', x - breakOffset);
88          breakLine2.setAttribute('y2', y + breakHeight);
89          breakLine2.setAttribute('class', 'number-line-break');
90          svg.appendChild(breakLine2);
91      }
92
93       function createJumpArrow(svg, x1, y1, x2, y2, jumpArcHeight, direction = 'forward',
           arrowSize = 5) { // Removed default color, use CSS
94          const path = document.createElementNS('http://www.w3.org/2000/svg', 'path');
95          const cx = (x1 + x2) / 2;
96          const cy = y1 - jumpArcHeight; // Arc is above the line
97          path.setAttribute('d', `M ${x1} ${y1} Q ${cx} ${cy} ${x2} ${y1}`); // Use y1 for
               end point to land on line
98          path.setAttribute('class', `jump-arrow`); // Rely on CSS for color
99          svg.appendChild(path);
100
101          // Arrowhead
102          const arrowHead = document.createElementNS('http://www.w3.org/2000/svg', 'path');
```

5

```javascript
        const dx = x2 - cx;
        const dy = y1 - cy; // Use y1 for angle calculation
        const angleRad = Math.atan2(dy, dx);
        let angleDeg = angleRad * (180 / Math.PI);
        arrowHead.setAttribute('class', 'jump-arrow-head'); // Rely on CSS for color

        if (direction === 'forward') {
            angleDeg += 180; // Point right
            arrowHead.setAttribute('d', 'M 0 0 L ${arrowSize} ${arrowSize/2} L ${
                arrowSize} ${-arrowSize/2} Z');
            arrowHead.setAttribute('transform', 'translate(${x2}, ${y1}) rotate(${
                angleDeg})');
        } else { // backward
            // angleDeg already points left-ish from Q curve end
            arrowHead.setAttribute('d', 'M 0 0 L ${-arrowSize} ${arrowSize/2} L ${-
                arrowSize} ${-arrowSize/2} Z'); // Pointy part is at (0,0)
             // We want to rotate to align with the curve's end direction
            arrowHead.setAttribute('transform', 'translate(${x2}, ${y1}) rotate(${
                angleDeg})');
        }
        svg.appendChild(arrowHead);
    }

    function drawStoppingPoint(svg, x, y, labelText, labelOffsetBase) {
        const circle = document.createElementNS('http://www.w3.org/2000/svg', 'circle');
        circle.setAttribute('cx', x);
        circle.setAttribute('cy', y);
        circle.setAttribute('r', 5);
        circle.setAttribute('class', 'stopping-point');
        svg.appendChild(circle);
        createText(svg, x, y + labelOffsetBase * 1.5, labelText, 'number-line-label');
    }
    // --- End Helper Functions ---

    // --- Main CBBO Automaton Function ---
    window.runCBBOAutomaton = function() {
        try {
            const minuend = parseInt(cbboMinuendInput.value);
            const subtrahend = parseInt(cbboSubtrahendInput.value); // Amount to subtract
            if (isNaN(minuend) || isNaN(subtrahend)) {
                outputElement.textContent = 'Please␣enter␣valid␣numbers␣for␣Minuend␣and␣
                    Subtrahend';
                diagramCBBOSVG.innerHTML = '';
                return;
            }
             if (subtrahend > minuend) {
                outputElement.textContent = 'Subtrahend␣cannot␣be␣greater␣than␣Minuend␣for␣
                    CBBO.';
                diagramCBBOSVG.innerHTML = '';
                return;
            }

            let output = '<h2>Counting Back by Bases and Ones (CBBO)</h2>\n\n';
            output += '<p><strong>Problem:</strong> ${minuend} - ${subtrahend}</p>\n\n';
```

6

```javascript
            const tensToSubtract = Math.floor(subtrahend / 10) * 10;
            const onesToSubtract = subtrahend % 10;

            output += `Step 1: Split subtrahend ${subtrahend} into ${tensToSubtract} + ${
                onesToSubtract}\n\n`;

            let currentVal = minuend;
            const tensSteps = [];
            if (tensToSubtract > 0) {
                output += 'Step 2: Count back by tens\n';
                for (let i = 10; i <= tensToSubtract; i += 10) {
                    tensSteps.push({ from: currentVal, to: currentVal - 10, action: '
                        Subtract 10' });
                    currentVal -= 10;
                }
                tensSteps.forEach(step => {
                    output += `<p>${step.from} - 10 = ${step.to}</p>\n`; // Simplified text
                });
                output += '\n';
            }

            const onesSteps = [];
            if (onesToSubtract > 0) {
                output += `Step ${tensToSubtract > 0 ? '3' : '2'}: Count back by ones\n`;
                for (let i = 1; i <= onesToSubtract; i++) {
                    onesSteps.push({ from: currentVal, to: currentVal - 1, action: '
                        Subtract 1' });
                    currentVal -= 1;
                }
                onesSteps.forEach(step => {
                    output += `<p>${step.from} - 1 = ${step.to}</p>\n`; // Simplified text
                });
                output += '\n';
            }

            const finalDifference = currentVal; // The final landing spot IS the
                difference
            output += `Result: ${minuend} - ${subtrahend} = ${finalDifference}`;
            outputElement.innerHTML = output;
            typesetMath();

            // Draw the diagram
            drawCBBONumberLineDiagram(diagramCBBOSVG, minuend, subtrahend, tensSteps,
                onesSteps, finalDifference);


        } catch (error) {
             console.error("Error in runCBBOAutomaton:", error);
            outputElement.textContent = `Error: ${error.message}`;
        }
    };
```

```
199    function drawCBBONumberLineDiagram(svg, minuend, subtrahend, tensSteps, onesSteps,
           finalDifference) {
200        if (!svg || typeof svg.setAttribute !== 'function') { return; }
201        svg.innerHTML = '';
202
203        const svgWidth = parseFloat(svg.getAttribute('width'));
204        const svgHeight = parseFloat(svg.getAttribute('height'));
205        const startX = 50;
206        const endX = svgWidth - 50;
207        const numberLineY = svgHeight / 2; // Center vertically
208        const tickHeight = 10;
209        const labelOffsetBase = 20;
210        const jumpHeight = 30; // Consistent jump height for CBBO
211        const jumpLabelOffset = 15;
212        const arrowSize = 5;
213        const scaleBreakThreshold = 40;
214
215        // Determine range for scaling
216        let diagramMin = finalDifference;
217        let diagramMax = minuend;
218
219        // Calculate scale and handle potential break (near 0, before diagramMin)
220        let displayRangeStart = diagramMin;
221        let scaleStartX = startX;
222        let drawScaleBreak = false;
223
224        if (diagramMin > scaleBreakThreshold) {
225            displayRangeStart = diagramMin - 10;
226            scaleStartX = startX + 30;
227            drawScaleBreak = true;
228            drawScaleBreakSymbol(svg, scaleStartX - 15, numberLineY);
229            drawTick(svg, startX, numberLineY, tickHeight);
230            createText(svg, startX, numberLineY + labelOffsetBase, '0', 'number-line-label
                   ');
231        } else {
232            displayRangeStart = 0;
233            drawTick(svg, startX, numberLineY, tickHeight);
234            createText(svg, startX, numberLineY + labelOffsetBase, '0', 'number-line-label
                   ');
235        }
236
237        const displayRangeEnd = diagramMax + 10;
238        const displayRange = Math.max(displayRangeEnd - displayRangeStart, 1);
239        const scale = (endX - scaleStartX) / displayRange;
240
241        // Function to convert value to X coordinate
242        function valueToX(value) {
243            if (value < displayRangeStart && drawScaleBreak) { return scaleStartX - 10; }
244            const scaledValue = scaleStartX + (value - displayRangeStart) * scale;
245            return Math.max(scaleStartX, Math.min(scaledValue, endX));
246        }
247
248        // Draw the main visible segment of the number line
249         const mainLineStartX = valueToX(displayRangeStart);
```

```
250         const mainLineEndX = valueToX(displayRangeEnd);
251         const numberLine = document.createElementNS('http://www.w3.org/2000/svg', 'line')
                ;
252       numberLine.setAttribute('x1', mainLineStartX);
253       numberLine.setAttribute('y1', numberLineY);
254       numberLine.setAttribute('x2', mainLineEndX);
255       numberLine.setAttribute('y2', numberLineY);
256       numberLine.setAttribute('class', 'number-line-tick');
257       svg.appendChild(numberLine);
258
259       // Add arrowhead to the right end
260       const mainArrowHead = document.createElementNS('http://www.w3.org/2000/svg', '
                path');
261       mainArrowHead.setAttribute('d', `M ${mainLineEndX - arrowSize} ${numberLineY -
                arrowSize/2} L ${mainLineEndX} ${numberLineY} L ${mainLineEndX - arrowSize} $
                {numberLineY + arrowSize/2} Z`);
262       mainArrowHead.setAttribute('class', 'number-line-arrow');
263       svg.appendChild(mainArrowHead);
264
265
266       // Draw Ticks and Labels
267       function drawTickAndLabel(value, index) {
268           const x = valueToX(value);
269           if (x < scaleStartX - 5 && value !== 0) return;
270
271           drawTick(svg, x, numberLineY, tickHeight);
272           const labelOffset = labelOffsetBase * (index % 2 === 0 ? 1 : -1.5); // Stagger
273           createText(svg, x, numberLineY + labelOffset, value.toString(), 'number-line-
                label');
274       }
275
276       // Collect all points to draw ticks for
277       let allPoints = new Set([minuend, finalDifference]); // Start and end
278       tensSteps.forEach(step => allPoints.add(step.to));
279       onesSteps.forEach(step => allPoints.add(step.to));
280       let sortedPoints = Array.from(allPoints).sort((a, b) => a - b);
281       let pointIndexMap = {};
282       let currentIndex = 0;
283       sortedPoints.forEach(point => {
284           if (point >= displayRangeStart || (point === 0 && !drawScaleBreak)) {
285               if (!(point < displayRangeStart && drawScaleBreak)) {
286                   pointIndexMap[point] = currentIndex++;
287                   drawTickAndLabel(point, pointIndexMap[point]);
288               }
289           }
290       });
291
292       // Draw tens jumps (Backward)
293       tensSteps.forEach((step, index) => {
294           const x1 = valueToX(step.from);
295           const x2 = valueToX(step.to);
296           if (x1 <= scaleStartX || x2 < scaleStartX) return; // Skip if outside visible
                range
297
```

```
298            const staggerOffset = index % 2 === 0 ? 0 : jumpHeight * 0.5;
299            createJumpArrow(svg, x1, numberLineY, x2, numberLineY, jumpHeight +
                   staggerOffset, 'backward', arrowSize);
300            createText(svg, (x1 + x2) / 2, numberLineY - (jumpHeight + staggerOffset) -
                   jumpLabelOffset, '-10', 'tens-jump-label');
301        });
302
303        // Draw ones jumps (Backward)
304        onesSteps.forEach((step, index) => {
305            const x1 = valueToX(step.from);
306            const x2 = valueToX(step.to);
307             if (x1 <= scaleStartX || x2 < scaleStartX) return; // Skip if outside visible
                   range
308
309            const staggerOffset = (tensSteps.length + index) % 2 === 0 ? 0 : jumpHeight *
                   0.5; // Continue staggering
310            createJumpArrow(svg, x1, numberLineY, x2, numberLineY, jumpHeight +
                   staggerOffset, 'backward', arrowSize);
311            createText(svg, (x1 + x2) / 2, numberLineY - (jumpHeight + staggerOffset) -
                   jumpLabelOffset, '-1', 'jump-label');
312        });
313
314        // Start point marker
315        if (valueToX(minuend) >= scaleStartX) {
316            drawStoppingPoint(svg, valueToX(minuend), numberLineY, 'Start',
                   labelOffsetBase);
317        }
318    }
319
320    function typesetMath() { /* Placeholder */ }
321
322    // Initial run on page load
323    runCBBOAutomaton();
324
325 });
326 </script>
327
328 </body>
329 <!-- New button for viewing PDF documentation -->
330 <button onclick="openPdfViewer()">Want to learn more about this strategy? Click here.</
       button>
331
332 <script>
333    function openPdfViewer() {
334        // Opens the PDF documentation for the strategy.
335        window.open('../PDF_Documentation_Of_Strategies/SAR_SUB_COBO.pdf', '_blank');
336    }
337 </script></html>
338 </html>
```

References

Carpenter, T. P., Fennema, E., Franke, M. L., Levi, L., & Empson, S. B. (1999). Children's mathematics: Cognitively guided instruction – videotape logs [supplementary material]. In *Children's mathematics: Cognitively guided instruction*. Heinemann, in association with The National Council of Teachers of Mathematics, Inc.

Hackenberg, A. (2025). *Course notes* [Unpublished course notes].