Subtraction Strategies: Rounding and Adjusting

Compiled by Theodore M. Savich

April 3, 2025

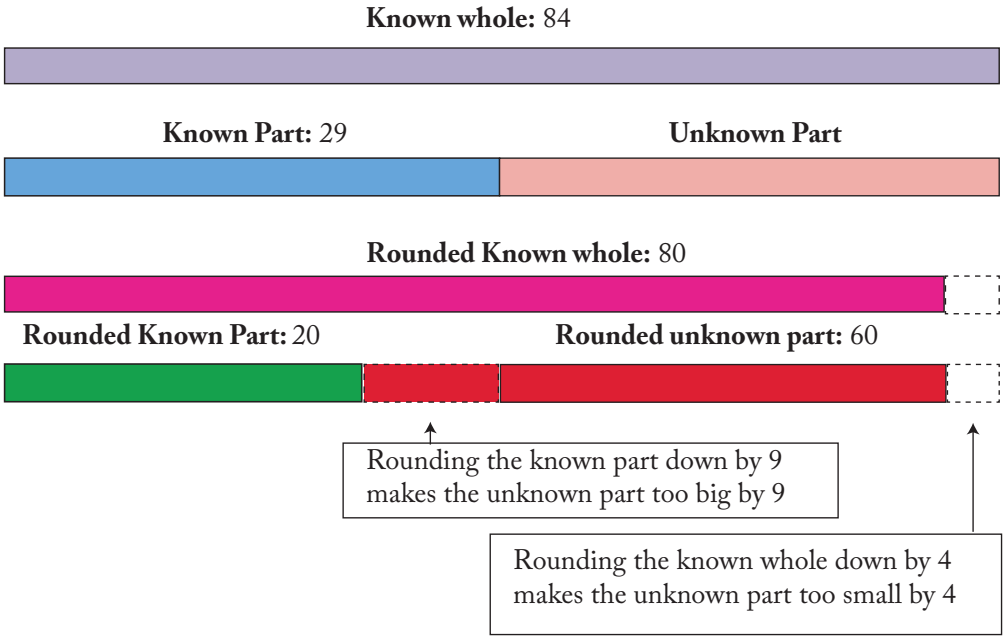
Rounding and Adjusting

Transcript

Video from Carpenter et al. (1999). Strategy descriptions and curation by Amy Hackenberg.

- **Teacher:** Kevin had 84 gumdrops. During the week, he ate 29 gumdrops. How many gumdrops does he have left?
- **Kevin:** 55.
- **Teacher:** How'd you get 55?
- **Kevin:** I knew if I had 80 gumdrops and I ate 20, I knew I would have 60 gumdrops. But I had to add 4 more because it was 84 minus 20, so that would be 64. And I took away 4 more, and that would be 60. But I had to take away 5 more and that would be 55.

Picture



Notation

Rounding

$$84 - 29 = \square \tag{1}$$
$$84 - 4 = 80 \tag{2}$$
$$29 - 9 = 20 \tag{3}$$
$$80 - 20 = 60 \tag{4}$$

Adjusting

$$60 + 4 = 64 \tag{5}$$
$$64 - 4 = 60 \tag{6}$$
$$60 - 5 = 55 \tag{7}$$

Explaining the Adjusting

- Kevin knew that if he had 80 gumdrops and ate 20, he would have 60 gumdrops left.
- Rounding the known whole down by 4 makes the unknown part too small by 4.
- So, adjust the difference by adding 4 gumdrops back to get 64.
- Rounding the known part down by 9 makes the unknown part too big by 9
- So, adjust the difference by subtracting 9. Kevin does this by chunking back by 4 (to get 60) and then by 5 (to get 55).

Description of Strategy

Change either the known part or the known whole to a "good" number—usually the nearest base—to make the subtraction easier. Then subtract and adjust your answer. This extra adjusting step can be a bit trickier than rounding when you add!

- If you round the known whole up, you pretend you had more than you really did, so the unknown part seems too big.
- If you round the known whole down, you act like you had less, and you'll need to add back what you subtracted at the end.
- Similarly, if you round the known part down, you're not subtracting enough and must add back in.
- If you round the known part up, you subtract too much and need to add some back to fix it.

Automaton Type

Pushdown Automaton (PDA): Needed to remember the amount of adjustment required.

Formal Description of the Automaton

We define the PDA as the 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_{0/accept}, Z_0, F)$$

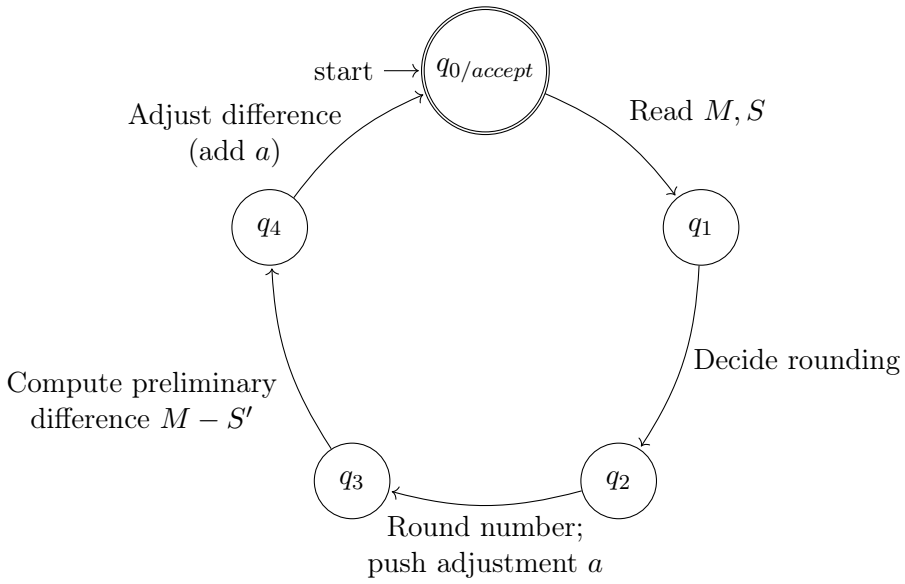
where:

- $Q = \{q_{0/accept},\ q_1,\ q_2,\ q_3,\ q_4\}$ is the set of states.
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is the input alphabet (representing the digits of the minuend M and subtrahend S).
- $\Gamma = \{Z_0\} \cup \{x \mid x \in \mathbb{Z}\}$ is the stack alphabet, where Z_0 is the initial stack symbol and x represents the adjustment value.
- $q_{0/accept}$ is the start state, which is also the accept state.
- $F = \{q_{0/accept}\}$ is the set of accepting states.

The transition function δ is defined by:

1. $\delta(q_{0/accept}, \text{“}M, S\text{”}, Z_0) = \{(q_1,\ Z_0)\}$
   (Read the minuend M and subtrahend S.)
2. $\delta(q_1,\ \varepsilon,\ Z_0) = \{(q_2,\ Z_0)\}$
   (Decide which number to round and determine the rounding strategy.)
3. $\delta(q_2,\ \varepsilon,\ Z_0) = \{(q_3,\ a\, Z_0)\}$
   (Perform the rounding. Let a be the adjustment amount where, for example, if rounding the subtrahend, $a = S' - S$.)
4. $\delta(q_3,\ \varepsilon,\ a) = \{(q_4,\ a)\}$
   (Compute the preliminary difference using the rounded value; that is, compute $M - S'$.)
5. $\delta(q_4,\ \varepsilon,\ a) = \{(q_{0/accept},\ Z_0)\}$
   (Adjust the preliminary difference by incorporating a (i.e., final difference $= (M - S') + a$) and output the result.)

Automaton Diagram for Rounding and Adjusting



HTML Implementation

```
<!DOCTYPE html>
<html>
<head>
<title>Subtraction Rounding and Adjusting</title>
<style>
  body { font-family: sans-serif; }
  #diagramSVG { border: 1px solid #d3d3d3; } /* Style SVG like canvas */
  #outputContainer { margin-top: 20px; }
  .diagram-label { font-size: 14px; }
</style>
    <script>
        MathJax = {
          tex: {
            inlineMath: [['$', '$'], ['\\(', '\\)']]
          },
          svg: {
            fontCache: 'global'
          }
        };
        </script>
        <script type="text/javascript" id="MathJax-script" async
          src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-svg.js">
        </script>
</head>
<body>

  <h1>Subtraction Strategies: Rounding and Adjusting</h1>

  <div>
    <label for="roundSubMinuend">Minuend (Whole):</label>
```

```
    <input type="number" id="roundSubMinuend" value="84">
  </div>
  <div>
    <label for="roundSubSubtrahend">Subtrahend (Part):</label>
    <input type="number" id="roundSubSubtrahend" value="29">
  </div>

  <button onclick="runSubtractionRoundingAutomaton()">Calculate and Visualize</button>

  <div id="outputContainer">
    <h2>Explanation:</h2>
    <div id="subRoundingOutput">
      <!-- Text output will be displayed here -->
    </div>
  </div>

  <h2>Diagram:</h2>
  <svg id="diagramSVG" width="600" height="450"></svg> <!-- Changed to SVG -->

  <script>
    document.addEventListener('DOMContentLoaded', function() {
    const outputElement = document.getElementById('subRoundingOutput');
    const minuendInput = document.getElementById('roundSubMinuend');
    const subtrahendInput = document.getElementById('roundSubSubtrahend');
    const diagramSVG = document.getElementById('diagramSVG'); // Get SVG element

    if (!outputElement || !diagramSVG) {
        console.warn("Element␣subRoundingOutput␣or␣diagramSVG␣not␣found");
        return;
    }

    window.runSubtractionRoundingAutomaton = function() {
        try {
            const minuend = parseInt(minuendInput.value);
            const subtrahend = parseInt(subtrahendInput.value);

            if (isNaN(minuend) || isNaN(subtrahend)) {
                outputElement.textContent = "Please␣enter␣valid␣numbers␣for␣minuend␣and␣
                    subtrahend.";
                return;
            }

            let output = '';
            output += `<h2>Rounding and Adjusting Subtraction</h2>`;
            output += `<p><strong>Original Problem:</strong> ${minuend} - ${subtrahend}</p
                >`; // MathJax in text output

            // Determine rounding strategy (round subtrahend down to nearest lower
                multiple of 10)
            const roundedSubtrahend = Math.floor(subtrahend / 10) * 10;
            const adjustment = subtrahend - roundedSubtrahend;

            output += `<p><strong>Step 1: Round Subtrahend Down</strong></p>`;
            output += `<p>Original Subtrahend: ${subtrahend}</p>`;
```

```
            output += `<p>Rounded Subtrahend: ${roundedSubtrahend}</p>`;
            output += `<p>Adjustment (amount subtracted): ${adjustment}</p>`;

            // Perform subtraction with rounded subtrahend
            const intermediateResult = minuend - roundedSubtrahend;

            output += `<p><strong>Step 2: Subtract Rounded Subtrahend</strong></p>`;
            output += `<p>${minuend} - ${roundedSubtrahend} = ${intermediateResult}</p>`;
                // MathJax in text output

            // Apply adjustment
            const finalResult = intermediateResult + adjustment;

            output += `<p><strong>Step 3: Apply Adjustment (Add back the subtracted amount
                )</strong></p>`;
            output += `<p>Preliminary Difference: ${intermediateResult}</p>`;
            output += `<p>Adjustment to add: ${adjustment}</p>`;
            output += `<p>Final Difference: ${intermediateResult} + ${adjustment} = ${
                finalResult}</p>`; // MathJax in text output

            // Final result
            output += `<p><strong>Result: ${minuend} - ${subtrahend} = ${finalResult}</
                strong></p>`; // MathJax in text output

            outputElement.innerHTML = output;
            typesetMath(); // Keep typesetMath for potential formatting

            // Draw the length diagram on the SVG
            drawLengthDiagram('diagramSVG', minuend, subtrahend, roundedSubtrahend,
                adjustment);


        } catch (error) {
            outputElement.textContent = `Error: ${error.message}`;
        }
    };

    function typesetMath() {
        if (window.MathJax && window.MathJax.Hub) {
            MathJax.Hub.Queue(["Typeset", MathJax.Hub]);
        }
    }

    function drawLengthDiagram(svgId, originalWhole, knownPart, roundedKnownPart,
        adjustment) {
        const svg = document.getElementById(svgId);
        if (!svg) return;

        // Clear SVG content
        svg.innerHTML = '';

        const svgWidth = parseFloat(svg.getAttribute('width'));
        const svgHeight = parseFloat(svg.getAttribute('height'));
        const barHeight = 30;
```

```
        const barSpacing = 60; // Increased barSpacing for more vertical space
        const textOffset = 5;
        const labelYOffset = -15; // Offset for labels above bars
        const scaleFactor = (svgWidth - 100) / originalWhole; // Scale to fit, with
            padding on sides
        let currentY = 50; // Increased starting Y position

        const colors = {
            knownWhole: '#D8D8D8', // Light grey
            knownPart: '#ADD8E6', // Light blue
            unknownPart: '#FFA07A', // Light Salmon
            roundedKnownWhole: '#D87093', // RosyBrown
            roundedKnownPart: '#90EE90', // Light Green
            roundedUnknownPart: '#FFD700' // Gold
        };

        // --- Initial Diagram ---

        // Known Whole (Minuend)
        const knownWholeRectWidth = originalWhole * scaleFactor;
        createText(svg, 50 + knownWholeRectWidth / 2, currentY + labelYOffset, `Known
            whole: ${originalWhole}`); // Centered label
        createRect(svg, 50, currentY, knownWholeRectWidth, barHeight, colors.knownWhole);


        // Known Part (Subtrahend) and Unknown Part
        const knownPartRectWidth = knownPart * scaleFactor;
        createText(svg, 50 + knownPartRectWidth / 2, currentY + barSpacing + labelYOffset,
             `Known Part: ${knownPart}`); // Centered label
        createRect(svg, 50, currentY + barSpacing, knownPartRectWidth, barHeight, colors.
            knownPart);
        const initialUnknownPart = originalWhole - knownPart;
        const unknownPartRectWidth = initialUnknownPart * scaleFactor;
        createRect(svg, 50 + knownPartRectWidth, currentY + barSpacing,
            unknownPartRectWidth, barHeight, colors.unknownPart);
        createText(svg, 50 + knownPartRectWidth + unknownPartRectWidth / 2, currentY +
            barSpacing + labelYOffset, `Unknown Part`); // Centered label


        currentY += 2 * barSpacing + 2 * barHeight + 30; // Increased spacing before
            rounded section

        // --- Rounded Diagram ---
        // Removed background rectangle for rounded section

        // Rounded Known whole
        const roundedKnownWholeRectWidth = originalWhole * scaleFactor;
        createText(svg, 50 + roundedKnownWholeRectWidth / 2, currentY + labelYOffset, `
            Rounded Known whole: ${originalWhole}`); // Centered label
        createRect(svg, 50, currentY, roundedKnownWholeRectWidth, barHeight, colors.
            roundedKnownWhole);


        // Rounded Known Part and Rounded Unknown part - Adjusted Label Y positions
```

```
        const roundedKnownPartRectWidth = roundedKnownPart * scaleFactor;
        createText(svg, 50 + roundedKnownPartRectWidth / 2, currentY + barSpacing +
            labelYOffset, `Rounded Known Part: ${roundedKnownPart}`); // Centered label
        createRect(svg, 50, currentY + barSpacing, roundedKnownPartRectWidth, barHeight,
            colors.roundedKnownPart);
        const roundedUnknownPart = originalWhole - roundedKnownPart;
        const roundedUnknownPartRectWidth = roundedUnknownPart * scaleFactor;
        createRect(svg, 50 + roundedKnownPartRectWidth, currentY + barSpacing,
            roundedUnknownPartRectWidth, barHeight, colors.roundedUnknownPart);
        createText(svg, 50 + roundedKnownPartRectWidth + roundedUnknownPartRectWidth / 2,
            currentY + barSpacing + labelYOffset, `Rounded unknown part: ${
            roundedUnknownPart}`); // Centered label


        // Adjustment Arrow and Text
        // createArrow(svg, 50 + roundedKnownPartRectWidth, currentY + barSpacing +
            barHeight + 5, 50 + roundedKnownPartRectWidth, currentY + 2 * barSpacing +
            barHeight + 35);
        createText(svg, 50 + roundedKnownPartRectWidth + 10, currentY +barSpacing +
            barHeight + 25, `Rounding the known part down by ${adjustment}`);
        createText(svg, 50 + roundedKnownPartRectWidth + 10, currentY +barSpacing +
            barHeight + 45, `makes the unknown part too big by ${adjustment}`);


    }

    // --- SVG Helper Functions ---
    function createRect(svg, x, y, width, height, fill, stroke = true) {
        const rect = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
        rect.setAttribute('x', x);
        rect.setAttribute('y', y);
        rect.setAttribute('width', width);
        rect.setAttribute('height', height);
        rect.setAttribute('fill', fill);
        if (stroke) {
            rect.setAttribute('stroke', 'black');
            rect.setAttribute('stroke-width', '1');
        }
        svg.appendChild(rect);
        return rect;
    }

    function createText(svg, x, y, textContent) {
        const text = document.createElementNS("http://www.w3.org/2000/svg", 'text');
        text.setAttribute('x', x);
        text.setAttribute('y', y);
        text.setAttribute('class', 'diagram-label');
        text.setAttribute('text-anchor', 'middle'); // Center text
        text.textContent = textContent;
        svg.appendChild(text);
        return text;
    }

    function createArrow(svg, x1, y1, x2, y2) {
```

```
        const line = document.createElementNS("http://www.w3.org/2000/svg", 'line');
        line.setAttribute('x1', x1);
        line.setAttribute('y1', y1);
        line.setAttribute('x2', x2);
        line.setAttribute('y2', y2);
        line.setAttribute('stroke', 'black');
        line.setAttribute('stroke-width', '1');

        const arrowHead = document.createElementNS("http://www.w3.org/2000/svg", 'path');
        const arrowSize = 5;
        arrowHead.setAttribute('d', `M ${x2} ${y2} L ${x2 - arrowSize} ${y2 - arrowSize} L
             ${x2 + arrowSize} ${y2 - arrowSize} Z`);
        arrowHead.setAttribute('fill', 'black');

        svg.appendChild(line);
        svg.appendChild(arrowHead);
    }

    function drawStoppingPoint(svg, x, y, labelText, labelOffsetBase = 20, index = 0) {
            const circle = document.createElementNS('http://www.w3.org/2000/svg', 'circle'
                );
            circle.setAttribute('cx', x);
            circle.setAttribute('cy', y);
            circle.setAttribute('r', 4);
            circle.setAttribute('class', 'stopping-point');
            svg.appendChild(circle);

            // Use the provided y parameter instead of numberLineY
            if (labelText) {
                // Add staggering based on index to prevent overlap with large values
                const labelOffset = labelOffsetBase * (index % 2 === 0 ? 1.5 : -1.8);
                createText(svg, x, y + labelOffset, labelText, 'number-line-label');
            }
        }


});
  </script>

  <!-- New button for viewing PDF documentation -->
<button onclick="openPdfViewer()">Want to learn more about this strategy? Click here.</
    button>

<script>
    function openPdfViewer() {
        // Opens the PDF documentation for the strategy.
        window.open('../SAR_SUB_Rounding.pdf', '_blank');
    }
</script>

</body>
</html>
```