Addition Strategies: Rounding and Adjusting

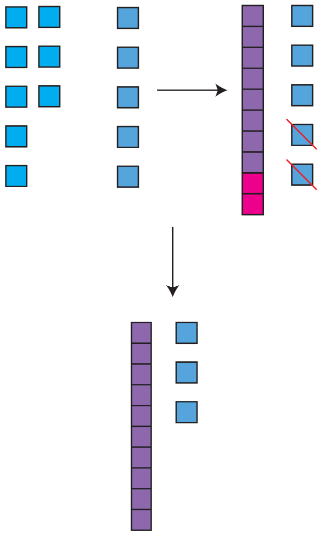
Compiled by: Theodore M. Savich

March 11, 2025

Transcript

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

- **Teacher:** Lucy has eight fish. She wants to buy five more fish. How many fish will Lucy have then?
- **Robert:** 13
- **Teacher:** How'd you get the 13?
- **Robert:** I just took the eight out. And then I, if she had ten fish, it would have been 15. If she had nine fish, it would have been 14. And if it would have been eight fish, which it was, it would have been 13. So, I just got 13.
- **Teacher:** Did you use those blocks to solve this problem?
- **Robert:** Well, I only used eight. I didn't use the other five, though. I used part of it in here (gestures to the mat with the blocks) and part of it in my head. you get 13.



Notation Representing Robert's Solution:

$$
\begin{aligned}
8+5 &= \square \\
8+2 &= 10 \\
10+5 &= 15 \\
8+5 &= 15-2 \\
8+5 &= 13
\end{aligned}
$$

Description of Strategy:

Objective: Rounding for Simplicity: We start by changing at least one number to a "friendlier" value — usually rounding it to the closest whole number of bases. For instance, if a number is just a few ones short of a multiple of 10, we can round it up so that it becomes exactly that multiple. This makes the arithmetic easier because we have well-known patterns (adding a full group of 10, for example) where the ones digit remains unchanged and only the tens (or "base") digit increases.

1. **The Need to Adjust:** When you round up a number, you are effectively adding a little extra to it. As a result, when you solve the simplified problem, your computed sum is slightly too high compared to the original one. To correct for this, you must subtract the extra amount that you added. Conversely, if you had rounded down (i.e., subtracted some value to simplify the number), then your computed sum would be too low, and you would need to add that amount back in.

2. **Why the Inverse Operation?** The principle is simple: whatever operation you use to alter the number for ease of calculation must be undone by the inverse operation to return to the original value.

   - If you **add** to round a number up, you must **subtract** later to adjust.
   - If you **subtract** to round a number down, you must **add** back the same amount after solving.

   This two-step process—first simplifying via rounding and then adjusting—helps you manage complex addition while keeping the final answer accurate to the original numbers.

Rounding and Adjusting

Description of Strategy

- **Objective:** Round one addend to a convenient number (usually a base multiple), perform the addition, then adjust the result.
- **Example:** $46+37$
  - Round 46 up to 50 (adding 4).
  - Add: $50+37=87$.
  - Adjust: Subtract the 4 added earlier: $87-4=83$.

Automaton Type

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

Automaton Description

- **States:**
  1. q_0: Start state.
  2. q_1: Read inputs and decide which number to round.
  3. q_2: Round the chosen number.
  4. q_3: Compute the adjustment.
  5. q_4: Perform the addition with the rounded number.
  6. q_5: Adjust the sum.
  7. q_{accept}: Accept state; output the final result.
- **Transitions:**
  - $q_0 \to q_1$: Read A and B; decide to round A.
  - $q_1 \to q_2$: Round A to A'.
  - $q_2 \to q_3$: Calculate adjustment $D = A' - A$.
  - $q_3 \to q_4$: Add A' and B.
  - $q_4 \to q_5$: Adjust the sum by subtracting D.
  - $q_5 \to q_{accept}$: Output the adjusted sum.

We define the PDA

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where:

- $Q = \{q_0,\, q_1,\, q_2,\, q_3,\, q_4,\, q_5,\, q_{\text{accept}}\}$ is the set of states.
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +\}$ is the input alphabet (for example, representing inputs like "46+37").
- $\Gamma = \{Z_0\} \cup \{x \mid x \in \mathbb{Z}\}$ is the stack alphabet, where Z_0 is the initial stack symbol and an integer x represents the adjustment amount.
- q_0 is the start state.
- Z_0 is the initial stack symbol.
- $F = \{q_{\text{accept}}\}$ is the set of accepting states.

The transition function

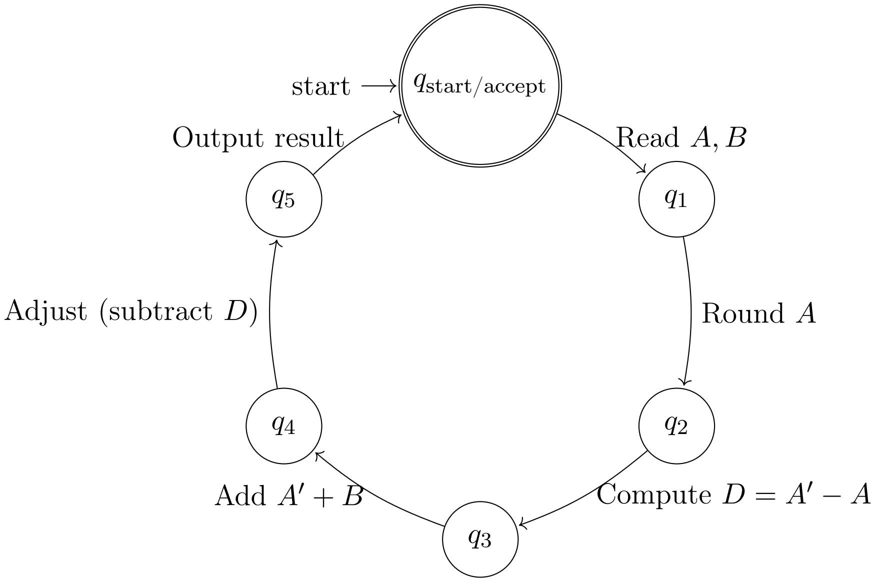
$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$$

is defined by the following transitions:

1. $\delta\Big(q_0,\, \text{“}A, B\text{”},\, Z_0\Big) = \{(q_1,\, Z_0)\}$.

2. $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$ (Round A to A').
3. $\delta(q_2, \varepsilon, Z_0) = \{(q_3, D\, Z_0)\}$ (Compute $D = A' - A$ and push D onto the stack).
4. $\delta(q_3, \varepsilon, D) = \{(q_4, D)\}$ (Perform the addition $A' + B$).
5. $\delta(q_4, \varepsilon, D) = \{(q_5, \varepsilon)\}$ (Adjust the sum by subtracting D; pop D from the stack).
6. $\delta(q_5, \varepsilon, Z_0) = \{(q_{\text{accept}}, Z_0)\}$ (Output the final result).

Automaton Diagram for Rounding and Adjusting



HTML Implementation

```
<!DOCTYPE html>
<html>
<head>
    <title>Addition Strategies: Rounding and Adjusting</title>
    <style>
        body { font-family: sans-serif; }
        #diagramRASVG { border: 1px solid #d3d3d3; }
        #outputContainer { margin-top: 20px; }
        .diagram-label { font-size: 12px; display: block; margin-bottom: 5px; }
    </style>
</head>
<body>

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

    <div>
        <label for="roundAddend1">Addend 1:</label>
        <input type="number" id="roundAddend1" value="46">
    </div>
    <div>
        <label for="roundAddend2">Addend 2:</label>
        <input type="number" id="roundAddend2" value="37">
    </div>

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

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

    <h2>Diagram:</h2>
    <svg id="diagramRASVG" width="100%" height="100%" viewBox="0␣0␣400␣700"
        preserveAspectRatio="xMidYMid␣meet"></svg>

    <script>
document.addEventListener(’DOMContentLoaded’, function() {
    const outputDiv = document.getElementById(’roundingOutput’);
    const roundAddend1Input = document.getElementById(’roundAddend1’);
    const roundAddend2Input = document.getElementById(’roundAddend2’);
    const diagramRASVG = document.getElementById(’diagramRASVG’);

    if (!outputDiv || !diagramRASVG) {
        console.warn("Element␣roundingOutput␣or␣diagramRASVG␣not␣found");
        return;
    }

    window.runRoundingAutomaton = function() {
        try {
            let a1 = parseInt(roundAddend1Input.value);
            let a2 = parseInt(roundAddend2Input.value);

            if (isNaN(a1) || isNaN(a2)) {
                outputDiv.textContent = "Please␣enter␣valid␣numbers␣for␣both␣addends";
                return;
            }

            let steps = ’’;
            steps += ’Initial␣Addends:␣’ + a1 + ’␣+␣’ + a2 + ’<br>’;

            // Decide which addend to round (round the first addend for simplicity)
            let remainderA1 = a1 % 10;
            let adjustmentA1 = remainderA1 === 0 ? 0 : 10 - remainderA1;
            let roundedA1 = a1 + adjustmentA1;
            let preliminarySum = roundedA1 + a2;
            let finalSum = preliminarySum - adjustmentA1;

            steps += ’Rounded␣’ + a1 + ’␣up␣to␣’ + roundedA1 + ’␣(added␣’ + adjustmentA1 +
                 ’)<br>’;
            steps += ’Preliminary␣Sum:␣’ + roundedA1 + ’␣+␣’ + a2 + ’␣=␣’ + preliminarySum
                 + ’<br>’;
            steps += ’Adjusting␣by␣subtracting␣’ + adjustmentA1 + ’␣(removing␣’ +
                adjustmentA1 + ’␣block’ + (adjustmentA1 > 1 ? ’s’ : ’’) + ’)<br>’;
            steps += 'Final␣Sum:␣' + preliminarySum + '␣-␣' + adjustmentA1 + '␣=␣' + finalSum;

            outputDiv.innerHTML = steps;
            typesetMath();

            // Draw the diagram
            drawRoundingAdjustingDiagram('diagramRASVG', a1, a2, roundedA1, adjustmentA1, preliminarySum, finalSum);

        } catch (error) {
            outputDiv.textContent = 'Error:␣' + error.message;
        }
    };

    function drawRoundingAdjustingDiagram(svgId, addend1, addend2, roundedAddend1, adjustment, preliminarySum, finalSum) {
        const svg = document.getElementById(svgId);
        if (!svg) return;
        svg.innerHTML = ''; // Clear SVG

        // Use a more compact layout
        const blockUnitSize = 8;
        const tenBlockWidth = blockUnitSize;
        const tenBlockHeight = blockUnitSize * 10;
        const blockSpacing = 2;
        const sectionSpacingY = 40;
        const startX = 20;
        let currentY = 30;

        // --- Original Addends (Side-by-Side) ---
        createText(svg, startX, currentY, `Original Addends: ${addend1} + ${addend2}`);
        currentY += 18;

        // Addend 1 Blocks
        let currentX1 = startX;
        let addend1_tens = Math.floor(addend1 / 10);
        let addend1_ones = addend1 % 10;
        let addend1Width = 0;

        for (let i = 0; i < addend1_tens; i++) {
            drawTenBlock(svg, currentX1, currentY, tenBlockWidth, tenBlockHeight, 'lightblue');
            currentX1 += tenBlockWidth + blockSpacing;
        }
        for (let i = 0; i < addend1_ones; i++) {
            drawBlock(svg, currentX1, currentY + tenBlockHeight - blockUnitSize, blockUnitSize, blockUnitSize, 'lightblue');
            currentX1 += blockUnitSize + blockSpacing;
        }
        addend1Width = currentX1 - startX;

        // Addend 2 Blocks - Positioned to the right of Addend 1
        let currentX2 = startX + addend1Width + 30;
        let addend2_tens = Math.floor(addend2 / 10);
        let addend2_ones = addend2 % 10;

        for (let i = 0; i < addend2_tens; i++) {
            drawTenBlock(svg, currentX2, currentY, tenBlockWidth, tenBlockHeight, 'lightcoral');
            currentX2 += tenBlockWidth + blockSpacing;
        }
        for (let i = 0; i < addend2_ones; i++) {
            drawBlock(svg, currentX2, currentY + tenBlockHeight - blockUnitSize, blockUnitSize, blockUnitSize, 'lightcoral');
            currentX2 += blockUnitSize + blockSpacing;
        }

        currentY += tenBlockHeight + sectionSpacingY;

        // --- Preliminary Sum (Rounded Addend 1 + Addend 2) ---
        createText(svg, startX, currentY, `Preliminary Sum: ${roundedAddend1} + ${addend2}`);
        currentY += 18;

        // Rounded Addend 1 Blocks (Light Green)
        let currentXRoundedA1 = startX;
        let roundedA1_tens = Math.floor(roundedAddend1 / 10);
        let roundedA1_ones = roundedAddend1 % 10;
        for (let i = 0; i < roundedA1_tens; i++) {
            drawTenBlock(svg, currentXRoundedA1, currentY, tenBlockWidth, tenBlockHeight, 'lightgreen');
            currentXRoundedA1 += tenBlockWidth + blockSpacing;
        }
        for (let i = 0; i < roundedA1_ones; i++) {
            drawBlock(svg, currentXRoundedA1, currentY + tenBlockHeight - blockUnitSize, blockUnitSize, blockUnitSize, 'lightgreen');
            currentXRoundedA1 += blockUnitSize + blockSpacing;
        }

        // Addend 2 Blocks (Light Coral)
        let currentXA2 = currentXRoundedA1 + 15;
        let addend2_tens_reused = Math.floor(addend2 / 10);
        let addend2_ones_reused = addend2 % 10;
        for (let i = 0; i < addend2_tens_reused; i++) {
            drawTenBlock(svg, currentXA2, currentY, tenBlockWidth, tenBlockHeight, 'lightcoral');
            currentXA2 += tenBlockWidth + blockSpacing;
        }
        for (let i = 0; i < addend2_ones_reused; i++) {
            drawBlock(svg, currentXA2, currentY + tenBlockHeight - blockUnitSize, blockUnitSize, blockUnitSize, 'lightcoral');
            currentXA2 += blockUnitSize + blockSpacing;
        }

        currentY += tenBlockHeight + 18;

        // --- Adjustment Section: Show Removed Blocks ---
        createText(svg, startX, currentY, `Adjustment: Remove ${adjustment} block${adjustment > 1 ? 's' : ''}`);
        currentY += 18;
        let currentX_adjust = startX;
        for (let i = 0; i < adjustment; i++) {
            drawRemovedBlock(svg, currentX_adjust, currentY, blockUnitSize, blockUnitSize);
            currentX_adjust += blockUnitSize + blockSpacing;
        }
        currentY += blockUnitSize + sectionSpacingY/2;

        // --- Final Sum (Adjusted) ---
        createText(svg, startX, currentY, `Final Sum (Adjusted): ${finalSum}`);
        currentY += 18;
        let currentXFinal = startX;
        let finalSum_tens = Math.floor(finalSum / 10);
        let finalSum_ones = finalSum % 10;
        for (let i = 0; i < finalSum_tens; i++) {
            drawTenBlock(svg, currentXFinal, currentY, tenBlockWidth, tenBlockHeight, 'gold');
            currentXFinal += tenBlockWidth + blockSpacing;
        }
        for (let i = 0; i < finalSum_ones; i++) {
            drawBlock(svg, currentXFinal, currentY + tenBlockHeight - blockUnitSize, blockUnitSize, blockUnitSize, 'gold');
            currentXFinal += blockUnitSize + blockSpacing;
        }

        // --- Helper SVG drawing functions ---
        function drawBlock(svg, x, y, width, height, fill) {
            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);
            rect.setAttribute('stroke', 'black');
            rect.setAttribute('stroke-width', '1');
            svg.appendChild(rect);
        }

        function drawTenBlock(svg, x, y, width, height, fill) {
            const group = document.createElementNS("http://www.w3.org/2000/svg", 'g');
            const backgroundRect = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
            backgroundRect.setAttribute('x', x);
            backgroundRect.setAttribute('y', y);
            backgroundRect.setAttribute('width', width);
            backgroundRect.setAttribute('height', height);
            backgroundRect.setAttribute('fill', fill);
            backgroundRect.setAttribute('stroke', 'black');
            backgroundRect.setAttribute('stroke-width', '1');
            group.appendChild(backgroundRect);

            for (let i = 0; i < 10; i++) {
                const unitBlock = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
                unitBlock.setAttribute('x', x);
                unitBlock.setAttribute('y', y + i * blockUnitSize);
                unitBlock.setAttribute('width', blockUnitSize);
                unitBlock.setAttribute('height', blockUnitSize);
                unitBlock.setAttribute('fill', fill);
                unitBlock.setAttribute('stroke', 'lightgrey');
                unitBlock.setAttribute('stroke-width', '0.5');
                group.appendChild(unitBlock);
            }
            svg.appendChild(group);
        }

        function drawRemovedBlock(svg, x, y, width, height) {
            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', '#ffe6e6');
            rect.setAttribute('stroke', 'red');
            rect.setAttribute('stroke-width', '1');
            svg.appendChild(rect);

            // Draw diagonal cross to indicate removal
            const line1 = document.createElementNS("http://www.w3.org/2000/svg", 'line');
            line1.setAttribute('x1', x);
            line1.setAttribute('y1', y);
            line1.setAttribute('x2', x + width);
            line1.setAttribute('y2', y + height);
            line1.setAttribute('stroke', 'red');
            line1.setAttribute('stroke-width', '1');
            svg.appendChild(line1);

            const line2 = document.createElementNS("http://www.w3.org/2000/svg", 'line');
            line2.setAttribute('x1', x + width);
            line2.setAttribute('y1', y);
            line2.setAttribute('x2', x);
            line2.setAttribute('y2', y + height);
            line2.setAttribute('stroke', 'red');
            line2.setAttribute('stroke-width', '1');
            svg.appendChild(line2);
        }

        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', 'start');
            text.setAttribute('font-size', '12px');
            text.textContent = textContent;
            svg.appendChild(text);
        }
     }

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

 </body>
 </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].