Change-Making Exercise Report

Greedy, Exhaustive Enumeration, and Branch&Bound

Team: Abbas Abdallah Hasan Bazzi Bilal Zahraman

October 5, 2025

Abstract

This report implements and evaluates a suite of programs for the classical *making change* problem using euro denominations. We implement: a greedy baseline (Prg1), exhaustive enumeration both iteratively and recursively (Prg2, Prg3), file output for diff validation, reversed generation order (Prg3bis), an improving trace (Prg5), computing the best solution with minimized I/O (Prg6), and branch&bound with pruning (Prg7) including an order comparison (Prg7bis). We also add optional tasks: top-k best solutions and "best + two less-worst". Experiments on 12.35 with denominations $\{5, 2, 1, 0.50, 0.20, 0.10, 0.05\}$ confirm correctness and show large speedups with pruning.

1 Problem

Given an amount A (in cents) and denominations $D = \langle d_1 > d_2 > \cdots > d_n \rangle$ (in cents), find nonnegative integers $(k_1, \ldots, k_n)$ with

$$\sum_{i=1}^{n} k_i d_i = A.$$

The *cost* of a solution is the number of coins $C = \sum_i k_i$. We seek (i) all feasible solutions; (ii) one with minimal C.

Instance for evaluation.

$$A = 1235 \text{ cents } (= 12.35), \quad D = \{500, 200, 100, 50, 20, 10, 5\}.$$

2 Programs Implemented

All code operates in **integer cents** to avoid floating-point issues.

Prg1: Greedy
Take as many of each denomination in order $d_1 \to d_n$. Returns a solution (not always optimal for arbitrary systems).

Prg2: Exhaustive (iterative)
Non-recursive DFS using an explicit stack; enumerates all solutions and can count them.

Prg3: Exhaustive (recursive)
Classic recursive DFS that prints each solution as it is found; also records recursion statistics.

Prg3 continuation:
Write the full recursive listing to a file for later diff with an instructor-provided `CORRECT_` file.

Prg3bis: Reversed order
Change per-level coin-choice order from `max` $\to 0$ to $0 \to$ `max` to study enumeration effects.

Prg4: Collect then display
Accumulate all solutions (no I/O during search) and print later in the same order.

Prg5: Improving trace
Log to a file whenever a new best (lower coin count) solution appears; the file is a monotone improvement trace.

Prg6: Best only
Enumerate all solutions but keep only the current best in memory; report elapsed time and enumeration count.

Prg7: Branch&Bound (cut)
Maintain partial coin count; prune branches where partial count $\geq$ current best. Report nodes/calls/time.

Prg7bis: Order comparison
Compare pruning effectiveness for `max_first` (as Prg3) vs. `min_first` (as Prg3bis).

Optional 8:
Keep the top-k best (fewest coins) solutions, sorted by coin count.

Optional 8:
Return the global best plus the two "less-worst" solutions (largest coin counts).

3 Algorithms (condensed)

Iterative DFS (Prg2). State is (i, r, prefix) for index i and remainder r. For denomination d_i, try $k \in [0, \lfloor r/d_i \rfloor]$. Push child states iteratively until $r = 0$ (yield solution) or $i = n$ (dead end).

Recursive DFS (Prg3). Same decision tree as above using recursion. We used order $\lfloor r/d_i \rfloor \to 0$ for Prg3 and $0 \to \lfloor r/d_i \rfloor$ for Prg3bis.

Dynamic Programming target. We compute an optimal coin count for each amount $a \leq A$ with a 1D DP to know the minimum number of coins. This is used only for *early stopping* when enumerating (for verification).

Branch&Bound (Prg7). With current partial coin count c, if $c \geq c^*$ (incumbent best), prune. We compare two exploration orders that strongly affect pruning power.

4 Complexity

Let $n = |D|$, and B be the total number of feasible solutions (combinatorial in general).

- **Greedy:** $O(n)$.
- **DP target:** $O(A \cdot n)$ time, $O(A)$ memory.
- **Enumeration (Prg2/Prg3):** $O(B)$ generated solutions; node expansions in the decision tree are $\Theta(B)$ in the worst case. I/O can dominate runtime when printing each solution.

- **Branch&Bound:** Worst-case still $O(B)$, but with an effective cut the explored nodes can drop by orders of magnitude.

5 Experimental Setup

- Amount $A = 12.35$; denominations $D = \{5, 2, 1, 0.50, 0.20, 0.10, 0.05\}$ euros (converted to cents).
- Python 3, single thread. Console results reported below are from the user's run.
- Files produced in the run:
  - `RecursiveSolution.txt` (recursive full listing for diff)
  - `all_solutions.txt`, `all_solutions.csv` (iterative writer)
  - `ImproveIncrementally.txt` (Prg5 trace)
  - `metrics_summary.csv` (profiler summary)

6 Results

6.1 Greedy (Prg1)

Greedy: {5.00×2 + 2.00×1 + 0.20×1 + 0.10×1 + 0.05×1} (coins: 6), remainder: 0.00.

6.2 Enumeration counts (Prg2/Prg3)

Metric	Value
Total solutions (iterative)	266,724
Total solutions (recursive)	266,724
Recursive calls (Prg3)	12,607,231
Nodes / branch choices (Prg3)	12,607,230

6.3 Best solution via DP target

Best (fewest coins): {5.00×2 + 2.00×1 + 0.20×1 + 0.10×1 + 0.05×1} (coins: 6).

6.4 Prg3bis (reversed order) first results

1) {0.05×247} (coins: 247); 2) {0.10×1 + 0.05×245} (coins: 246); 3) {0.10×2 + 0.05×243} (coins: 245).

6.5 Prg4 (collect, then print)

Collected 266,724 solutions without printing during search; the first three match the Prg3 order.

6.6 Prg5 (improving trace)

Final best coin count: 6. Trace file: `ImproveIncrementally.txt`.

6.7 Prg6 (best only, timing)

Metric	Value
Best coin count	6
Solutions enumerated	266,724
Elapsed time	4.774428 s

6.8 Prg7 (branch&bound) and Prg7bis (order comparison)

Method/Order	Best cost	Nodes	Time (s)
BnB (max_first, Prg7)	6	77,609	0.005643
BnB (Prg7bis: max_first)	6	77,609	0.004979
BnB (Prg7bis: min_first)	6	3,776,302	0.397484

Observation. With the partial-cost cut, exploring larger-denomination counts first (`max_first`) prunes dramatically better on this instance.

6.9 Optional Program 8 (Top-5)

#	Solution	Coins
1	{5.00×2 + 2.00×1 + 0.20×1 + 0.10×1 + 0.05×1}	6
2	{5.00×2 + 2.00×1 + 0.20×1 + 0.05×3}	7
3	{5.00×2 + 2.00×1 + 0.10×3 + 0.05×1}	7
4	{5.00×2 + 1.00×2 + 0.20×1 + 0.10×1 + 0.05×1}	7
5	{5.00×2 + 2.00×1 + 0.10×2 + 0.05×3}	8

6.10 Optional Program 8 (Best + two less-worst)

- **Best:** {5.00×2 + 2.00×1 + 0.20×1 + 0.10×1 + 0.05×1} (coins: 6)
- Less-worst #1: {0.05×247} (coins: 247)
- Less-worst #2: {0.10×1 + 0.05×245} (coins: 246)

6.11 Profiler metrics summary (from `metrics_summary.csv`)

Algorithm	Time (s)	Peak Mem (KB)	Printed Chars
Greedy	0.000616	1.4	83
Exhaustive/Iter (print all)	40.913459	23,913.5	17,859,559
Exhaustive (store all)	37.908426	61,126.5	34
Early-Stop	0.009865	19.8	75
Branch&Bound (profiled)	0.137244	2.2	102

Note: The profiled BnB time includes measurement overhead (printing, tracemalloc); the raw Prg7 timing above shows the core search time (≈ 5.6 ms).

7 Discussion

I/O vs. computation. Printing every solution (Prg2/Prg3) dominates runtime. The profiler recorded ∼17.86 million printed characters for the full iterative listing, which explains the 40.9

s runtime. Prg6, which avoids per-solution I/O, is far faster despite enumerating the same 266k solutions.

Effectiveness of pruning. Prg7 cuts the search from $\sim$ 12.6M nodes (raw recursion) to $\sim$ 77k nodes, reducing runtime to milliseconds in the core algorithm.

Order matters. Prg7bis shows that `max_first` ordering greatly improves pruning with the partial-cost bound on this instance (77k vs. 3.78M nodes).

8 Reproducibility

1. Run the script: `python Tp.py`
2. Files produced: `RecursiveSolution.txt`, `all_solutions.txt`, `all_solutions.csv`, `ImproveIncrementall` `metrics_summary.csv`
3. (Optional) When a gold file is provided, diff the recursive listing: `diff CORRECT_RecursiveSolution.txt RecursiveSolution.txt`

Appendix A: First 5 Enumerated Solutions (Iterative)

```
1. {5.00x2 + 2.00x1 + 0.20x1 + 0.10x1 + 0.05x1} (coins: 6)
2. {5.00x2 + 2.00x1 + 0.20x1 + 0.05x3} (coins: 7)
3. {5.00x2 + 2.00x1 + 0.10x3 + 0.05x1} (coins: 7)
4. {5.00x2 + 2.00x1 + 0.10x2 + 0.05x3} (coins: 8)
5. {5.00x2 + 2.00x1 + 0.10x1 + 0.05x5} (coins: 9)
```

Repository

All code and materials are available at: https://github.com/abbass03/TP