

# Change-making problem

From Wikipedia, the free encyclopedia

The change-making problem addresses the following question: how can a given amount of money be made with the least number of coins of given denominations? It is a knapsack type problem, and has applications wider than just currency.

## Contents

- 1 Mathematical definition
- 2 Non-currency examples
- 3 Methods of solving
  - 3.1 Simple dynamic programming
    - 3.1.1 Optimal Substructure
    - 3.1.2 Implementation
  - 3.2 Dynamic programming with the probabilistic convolution tree
  - 3.3 Greedy method
- 4 Related problems
- 5 See also
- 6 References

## Mathematical definition

Coin values can be modeled by a set of  $n$  distinct positive integer values (whole numbers), arranged in increasing order as  $w_1 = 1$  through  $w_n$ . The problem is: given an amount  $W$ , also a positive integer, to find a set of non-negative (positive or zero) integers  $\{x_1, x_2, \dots, x_n\}$ , with each  $x_j$  representing how often the coin with value  $w_j$  is used, which minimize the total number of coins

$$\sum_{j=1}^n x_j$$

subject to

$$\sum_{j=1}^n w_j x_j = W.$$

## Non-currency examples

An application of change-making problem can be found in computing the ways one can make a nine dart finish in a game of darts.

## Methods of solving

## Simple dynamic programming

A classic dynamic programming strategy works upward by finding the combinations of all smaller values that would sum to the current threshold. Thus, at each threshold, all previous thresholds are potentially considered to work upward to the goal amount  $W$ . For this reason, this dynamic programming approach may require a number of steps that is at least quadratic in the goal amount  $W$ .

### Optimal Substructure

We can use a dynamic programming strategy to solve this problem because it exhibits optimal substructure.

Proof.

Suppose  $S$  is the optimal solution that contains exactly  $n$  coins. Then  $S' = S - c$ , for some  $c \in S$ , is an optimal solution for the subproblem that contains exactly  $n - c$  coins.

Now, suppose  $S'$  is not the optimal solution that contains  $n - c$  coins, then there must exist an optimal solution that we call  $X \neq S'$ . Since  $X$  is the optimal solution, it must contain less number of coins than  $S'$ .

If we combine  $X$  with  $c$ , we obtain the optimal solution that contains exactly  $n$  coins, but this contradicts our assumptions that  $S$  is the optimal solution for the original problem.

### Implementation

The following is a dynamic programming implementation (with Python 3) using a matrix for keeping track of optimal solutions to subproblems. At the end, it returns the minimum amount of coins. If you want to obtain the set of coins for the optimal solution, you can keep track of them using for example another matrix.

```

1 def _get_change_making_matrix(set_of_coins, r):
2     m = [[0 for _ in range(r + 1)] for _ in range(len(set_of_coins) + 1)]
3
4     for i in range(r + 1):
5         m[0][i] = i
6
7     return m
8
9 def change_making(coins, n):
10     """This function assumes that all coins are available infinitely.
11
12     n is the number that we need to obtain with the fewest number of coins.
13
14     coins is a list or tuple with the available denominations. """
15
16     m = _get_change_making_matrix(coins, n)
17
18     for c in range(1, len(coins) + 1):
19
20         for r in range(1, n + 1):
21
22             # Just use the coin coins[c - 1].
23             if coins[c - 1] == r:
24                 m[c][r] = 1
25
26             # coins[c - 1] cannot be included.
27             # We use the previous solution for making r,
28             # excluding coins[c - 1].

```

```

29     elif coins[c - 1] > r:
30         m[c][r] = m[c - 1][r]
31
32     # We can use coins[c - 1].
33     # We need to decide which one of the following solutions is the best:
34     # 1. Using the previous solution for making r (without using coins[c - 1]).
35     # 2. Using the previous solution for making r - coins[c - 1] (without using coins[c - 1]) plus this 1 ex
36     else:
37         m[c][r] = min(m[c - 1][r], 1 + m[c][r - coins[c - 1]])
38
39     return m[-1][-1]

```

## Dynamic programming with the probabilistic convolution tree

The probabilistic convolution tree<sup>[1]</sup> can also be used as a more efficient dynamic programming approach. The probabilistic convolution tree merges pairs of coins to produce all amounts which can be created by that pair of coins (with neither coin present, only the first coin present, only the second coin present, and both coins present), and then subsequently merging pairs of these merged outcomes in the same manner. This process is repeated until the final two collections of outcomes are merged into one, leading to a balanced binary tree with  $W \log(W)$  such merge operations. Furthermore, by discretizing the coin values, each of these merge operations can be performed via convolution, which can often be performed more efficiently with the fast Fourier transform (FFT). In this manner, the probabilistic convolution tree may be used to achieve a solution in sub-quadratic number of steps: each convolution can be performed in  $n \log(n)$ , and the initial (more numerous) merge operations use a smaller  $n$ , while the later (less numerous) operations require  $n$  on the order of  $W$ .

The probabilistic convolution tree-based dynamic programming method also efficiently solves the probabilistic generalization of the change-making problem, where uncertainty or fuzziness in the goal amount  $W$  makes it a discrete distribution rather than a fixed quantity, where the value of each coin is likewise permitted to be fuzzy (for instance, when an exchange rate is considered), and where different coins may be used with particular frequencies.

## Greedy method

For the so-called canonical coin systems, like the one used in US and many other countries, a greedy algorithm of picking the largest denomination of coin which is not greater than the remaining amount to be made will produce the optimal result.<sup>[2]</sup> This is not the case for arbitrary coin systems, though: if the coin denominations were 1, 3 and 4, then to make 6, the greedy algorithm would choose three coins (4, 1, 1) whereas the optimal solution is two coins (3, 3).

## Related problems




The "optimal denomination problem"<sup>[3]</sup> is a problem for people who design entirely new currencies: What denominations should be chosen for the coins in order to minimize the average cost of making change—i.e., the average number of coins needed to make change? The version of this problem assumed that the people making change will use the minimum number of coins (from the denominations available). One variation of this problem assumes that the people making change will use the "greedy algorithm" for making change,

even when that requires more than the minimum number of coins. Most current currencies use a 1-2-5 series, but some other set of denominations would require fewer denominations of coins or a smaller average number of coins to make change or both.

## See also

- List of knapsack problems
- Coin problem
- The coin collector's problem

## References

- Serang (2014): Serang, O. (2012). "The Probabilistic Convolution Tree: Efficient Exact Bayesian Inference for Faster LC-MS/MS Protein Inference" (<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3953406>). *PLOS ONE*. 9 (3): e91507. doi:10.1371/journal.pone.0091507 (<https://doi.org/10.1371%2Fjournal.pone.0091507>). PMC 3953406 (<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3953406>)  PMID 24626234 (<https://www.ncbi.nlm.nih.gov/pubmed/24626234>).
  - Xuan Cai (2009). "Canonical Coin Systems for CHANGE-MAKING Problems". *Proceedings of the Ninth International Conference on Hybrid Intelligent Systems*. 1: 499 – 504. doi:10.1109/HIS.2009.103 (<https://doi.org/10.1109%2FHIS.2009.103>).
  - J. Shallit. "What this country needs is an 18c piece" (<http://www.cs.uwaterloo.ca/~shallit/Papers/change2.pdf>) (PDF). *Mathematical Intelligencer*. 25 (2): 20 – 23. doi:10.1007/BF02984830 (<https://doi.org/10.1007%2FBF02984830>).
- X. Cai (2009). "Canonical Coin Systems for Change-Making Problems". *Proceedings of the Ninth International Conference on Hybrid Intelligent Systems*. 499 – 504. arXiv:0809.0400 (<https://arxiv.org/abs/0809.0400>)  doi:10.1109/HIS.2009.103 (<https://doi.org/10.1109%2FHIS.2009.103>).
  - M. Adamaszek, A. Niewiarowska (2010). "Combinatorics of the change-making problem". *European Journal of Combinatorics*. 31 (1): 47 – 63. arXiv:0801.0120 (<https://arxiv.org/abs/0801.0120>)  doi:10.1016/j.ejc.2009.05.002 (<https://doi.org/10.1016%2Fj.ejc.2009.05.002>).

Retrieved from "[https://en.wikipedia.org/w/index.php?title=Change-making\\_problem&oldid=783805023](https://en.wikipedia.org/w/index.php?title=Change-making_problem&oldid=783805023)"

Categories: Number theory | Recreational mathematics | Combinatorial optimization

- This page was last edited on 2017-06-05, at 03:36:57.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.