International Journal of Pure and Applied Mathematics
Volume 119 No. 10 2018, 625-630
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
Special Issue

ijpam.eu

On Some Aspects of the Greedy Algorithm for Change Making

[1]Janeena Shaju, [2]Jemimah Johnson Neelamkavil and [3]R. Nandakumar

[1]Department of Computer Science and IT,
Amrita School of Arts and Sciences, Kochi,
Amrita Vishwa Vidyapeetham, India.
janeenashaju@gmail.com

[2]Department of Computer Science and IT,
Amrita School of Arts and Sciences, Kochi,
Amrita Vishwa Vidyapeetham, India.
mimahjohnson@gmail.com

[3]Department of Computer Science and IT,
Amrita School of Arts and Sciences, Kochi,
Amrita Vishwa Vidyapeetham, India.
nanadacumar@gmail.com

Abstract

The change making problem is a classic NP-complete problem. It aims, given a coinage system, to partition a specified target amount, using the least possible number of the coins available. It is well known that for some coinage systems, such as for example, the Indian currency system, a very fast greedy algorithm achieves such a partition but for a general coinage, we might need to use dynamic programming which can yield only a pseudo-polynomial time algorithm.

Here, we examine some aspects of the greedy algorithm ('Greedy' hereafter) for change making. We try to characterize those a coinage for which greedy works ('good coinages'). We also examine coinages where it fails and try to further classify them as those systems which can be made 'good' by the addition of a finite number of new denominations and those which can't be.

1. Introduction

The Change-Making Problem can be stated as follows: we are given a coin system and want to partition a given target value 'j' using the fewest coins possible. Change Making is rich in variants which are easy to state and understand but probably difficult to settle–for instance, we could study the on 2-way change making, which expresses the target amount as the difference between two amounts (the payment and the change being returned) and to minimize the total number of coins used for the two amounts together; for example, to hand over 87 rupees in the Indian system, instead of the one-way problem which has the optimal answer: 87 = 50+20+10+5+2 (involving 5 coins), we have a better two-way answer: 87 = 100 – (10+2+1)-involving only 4 coins[1]. Here, we return to the one-way change making and study its basic Greedy solution.

For some coin systems such as the present Indian coin system (denominations: 1,2,5,10,20,50,100,….) a 'Greedy' algorithm that repeatedly takes the 'best' available coin will yield the optimal solution (e.g.: for target 37 with the Indian coinage, Greedy gives (20+10+5+2) which is indeed, the optimal breakup for the coinage).It is easily seen that for coin systems such as, say, (1,4,6,10…), Greedy fails–indeed, given a target amount 8, Greedy gives the breakup of (6+1+1) whereas (4+4) is clearly the optimal partition. For such coinages, this optimal coinage can be obtained by using dynamic programming (DP). The DP algorithm takes pseudo polynomial time[2].

Dynamic programming has very high computational complexity and Greedy is extremely fast (it only needs for the denominations in the coinage to be sorted once). So the problem of determining if for a given coinage, Greedy gives the optimal partition (we call such coinages 'good coinages') for any target amount becomes important – in other words, we ask if we can characterize a specified coinage as good or otherwise efficiently. A fast algorithm for this would be useful if a large number of large target amounts need to be changed.

2. Characterizing Good Coin Systems

To check if Greedy works (i.e. it yields the best partition) for a certain target amount, we could run both greedy and DP for that target and see if both give the same result. That brings up the basic question: is it possible to assert that if greedy is seen to work for all targets up to a certain target amount, it will work for all larger targets? This characterization is computationally expensive (because for various targets, both Greedy and DP needs to be run and their outputs compared) but as noted above, this can yield savings if a very large number of large targets will need to be partitioned.

Proposition 1: To check if a coinage is good, it is enough to check if for all target values up to and including twice the largest denomination in the coinage, greedy works (i.e. it gives the same answer as DP). If greedy works for all target

values up to this critical value, it will work for all larger target values.

Example: Consider the coinage with denominations: {1, 2, 4, 5, 8}. By repeatedly comparing Greedy and DP, we see that for this coinage, greedy works for any arbitrarily large target amount. Our claim above is that we need to check for targets only up to 16(Greedy gives the optimal partition 8+8), to assert that Greedy works for any larger target values.

Proof: We prove our proposition by contradiction.

Let the coinage be {1, a1, a2,.. a_m} where a_m is the largest denomination. By our initial assumption, Greedy gives an optimal partition for all target amounts up to and including $2a_m$.

Let us assume that the first target value for which Greedy fails is between $2a_m$ and $3a_m$. The first failure is at $2a_m + k$ where $k < a_m$. First we observe that if the denominations used in an optimal partition of any target W are grouped into 2 sets of coins, with the two sets adding to w1 and w2 respectively, then both 'subtargets' w1 and w2 will be necessarily, optimally partitioned.

We now assert that the optimal partition of $2a_m+k$ necessarily contains the denomination a_m. Indeed, if a_m is not in the partition, then any partition of $2a_m$ +k will consist of two subtargets such that one subtarget necessarily will have value between a_m and $2a_m$. By earlier observation, both these subtargets will need to be partitioned optimally. And since by assumption, Greedy works for targets up to $2a_m$, we will necessarily use a_m in the optimal breakup of that subtarget between a_m and $2a_m$. Thus, we are constrained to take at least one a_m in the optimal partition of $2a_m+k$. Let us do so (this is precisely the first step done by greedy on $2a_m+k$) and that will leave us with a subtarget within a_m and $2a_m$, which we need to partition optimally. Since for values up to $2a_m$, Greedy works by assumption, we use it for the subtarget and are led to the conclusion that Greedy has actually worked for $2a_m$ +k.

We have thus argued that for targets up to $3a_m$, Greedy will necessarily work. So the first target for which Greedy fails will be, if such a target exists, $3a_m + k$. Again, we see that a_m is necessarily used in the partition of this target and hence it cannot be a counter example at all.

By repeating the argument, we can prove the proposition for any higher range (n a_m, (n+1) a_m) thus proving it fully.

Remark: It could be computationally expensive to check all targets up to $2a_m$. This brings up the following question: can one identify a smaller set of targets to check before concluding whether a coinage is good?

We now present some specific ways in which a coin system can be good.

Proposition_2: If denominations are in an Arithmetic Progression beginning with 1, the coinage will be good.

Proof: Any pair of numbers {a, b, with a<=b} appearing in an AP can be replaced by another pair of elements {a', b'} in that AP with b' >b and a'<a. So, if the partition of any target features denominations a and b, we can replace them with a' and b' in the partition. Greedy will choose a' and b' that will not be inferior to choosing a and b.

We state without proof, two further observations:

Proposition 3: If the denominations are successive primes, it is good.

Proposition 4: If denominations are in a Geometric Progression starting with 1, the coinage will be good.

3. 'Correcting' Bad Coin System

We first examine a class of bad coinages which can be converted to good systems by adding a single extra denomination.

Example 1: For the system {1, 2, 4, 5}, the least target value for which Greedy fails is 8 (The optimal partition is 4+4 and Greedy returns 5+2+1). Now, if we add this lowest failure value to the coinage, the resulting system: {1, 2, 4, 5, 8} turns good.

Example 2: The system, {1, 2, 3, 5, 6} fails for the target 10. However, it can be corrected (made good) by adding the denomination 9 or 10.

We now state an inference from our experiments:

Inference: Any coinage of the form {1, 2, n, n+1} will be bad and can be made good by adding the lowest target for which it fails. Such bad coinages appear unique in the sense that they fail only for one single target value and can bemade good by adding that single value as denomination. Another example is {1, 2, 6, 7} which can be corrected by adding 12.

Example 3: The coinage {1, 3, 4} fails for the targets 6, 10, 14… an infinite arithmetic progression. This cannot be corrected by adding any finite number of new denominations. Indeed, if we add 6 to the coinage (6 being the lowest failure value), the failure set of {1, 3, 4, 6} becomes 8, 14, 20,… Adding 8 will lead to a fresh infinite sequence of failures and so on.

Example 4: The coinage {1, 3, 5, 6} fails for targets: 8,10,14,16, 20,22,…. These failure values show a 'basic cluster of two values {8, 10}' and an arithmetic progression of this cluster with the highest denomination (here 6) as the common difference. This coinage cannot be corrected by adding any finite number of additional denominations.

Question: Can one construct a coinage that fails for exactly n values where n is finite and greater than 1? Our conjecture based on experimentation, is that this is

unlikely.

4. Conclusion

We have examined some variants on the change making problems, presented some results and asked some further questions for future study. The question of verifying whether a given coinage is good in a computationally efficient manner remains open, as far as we know.

References

[1] David Mount, Design and Analysis of Algorithms–Lecture notes https://www.cs.umd.edu/~mount/451/Lects/451lects.pdf.

[2] Arun A.V., Anundh K.G., Nandakumar R., On some variants of the change making problem, Journal of Engineering and applied Sciences (2017).

[3] Kozen D., Zaks S., Optimal bounds for the change-making problem, Theoretical Computer Science 123(2) (1994), 377-388.

[4] Lueker G.S., Two NP-complete problems in nonnegative integer programming, Technical Report 178, Princeton University (1990).

[5] Chang S.K., Gill A., Algorithmic solution of the change making problem, J.Assoc. Computer. Mach. 17(1) (1970), 113-122.

[6] Garey M.R., Johnson, D.S., Computers and intractability: A guide to the theory of NP completeness (series of books in the mathematical sciences), ed. Computers and Intractability (1979).