CS 124, PROGRAMMING ASSIGNMENT 3

Chao Cheng, Matthew Qu
Due: April 28, 2020
`chaocheng@college.harvard.edu`
`matthewqu@college.harvard.edu`

First, let us give a dynamic programming solution to the number partition problem. Let $A = [a_1, a_2, \dots, a_n]$ be an array of n non-negative numbers that sums to b. Then, define the following function $f(r, i)$ that returns True if there exists a partition of the subarray $[a_1, a_2, \dots, a_i]$ such that the residue is r. Here, we have two bounds on r and i: we know that $1 \leq i \leq n$ and $0 \leq r \leq b$. For our base case, we realize that an array of one element can only have one possible residue. Therefore, $f(a_1, a_1)$ is True, while $f(c, a_1)$ is False for all $c \neq a_1$. Furthermore, we have the following recursion:

$$f(r, i) = f(|r - a_i|, i - 1) \vee f(r + a_i, i - 1).$$

(Note that $r + a_i$ is necessarily non-negative, although $r - a_i$ is not; therefore, we require the absolute value.) In other words, the residue r is achievable with the first i terms if either residue $|r \pm a_i|$ is achievable with the first $i - 1$ terms. This leads us to a rather straightforward solution polynomial in nb time: we simply create a $(b+1) \times n$ matrix where the entry in the i^{th} row and j^{th} column is the value of $f(i - 1, j)$. If this value is true, then we also keep track of the sequence of signs that yield the given residue. Then, by filling out the matrix column by column, we are guaranteed to calculate all previous values necessary for each recursion. Once we have filled out all values in our matrix, we find the first entry in the n^{th} column; this corresponds to the sequence S that yields the minimal residue. We see that this dynamic programming algorithm runs in $nb + n$ time, which is polynomial in nb.

We can implement the Karmarkar-Karp algorithm in $O(n \log n)$ time by using a max heap. Creating the heap takes $O(n)$ time, while extracting the largest two elements and inserting their difference each takes $O(\log n)$ time. Since we must extract and insert a total $n - 1$ times, the total runtime is $O((n - 1) \log n) + O(n) = O(n \log n)$, as desired.

Below is a table depicting the average runtime, average residue, and median residue over 100 trials for each of the seven algorithms.

Algorithm Residues and Runtimes			
Algorithm	*Avg. Runtime (s)*	*Avg. Residue*	*Median Residue*
Karmarkar-Karp	$9.0745 \cdot 10^{-4}$	$2.7362 \cdot 10^{5}$	$1.3538 \cdot 10^{5}$
Repeated Random	5.1839	$3.0404 \cdot 10^{8}$	$2.0646 \cdot 10^{8}$
Hill Climbing	0.75468	$2.6380 \cdot 10^{8}$	$1.6932 \cdot 10^{8}$
Simulated Annealing	2.1690	$2.9631 \cdot 10^{8}$	$1.8898 \cdot 10^{8}$
Prepartioned Repeated Random	22.040	207.59	164
Prepartioned Hill Climbing	19.147	194.33	156.5
Prepartioned Simulated Annealing	20.023	188.63	133.5

All of the runtimes were as expected. The Karmarkar-Karp algorithm is deterministic and only needs to run once, whereas the other algorithms ran for 25000 iterations; thus, it clearly ran the fastest. The standard variants of the three algorithms also ran faster than their prepartitioned counterparts; this is because the prepartitioned algorithms must use Karmarkar-Karp to calculate the residue instead of a simple sum. When comparing the runtimes for the three standard algorithms, we see that the random algorithm takes the longest time, which is expected because it must generate an entirely new solution each iteration. In contrast, the hill climbing algorithm only changes one or two indices of a solution at a time. The simulated annealing algorithm runs slower than hill climbing because it is identical except that it must generate an additional random number and calculate another expression every time the generated solution is worse. These results are analogous for the prepartioned variants as well.

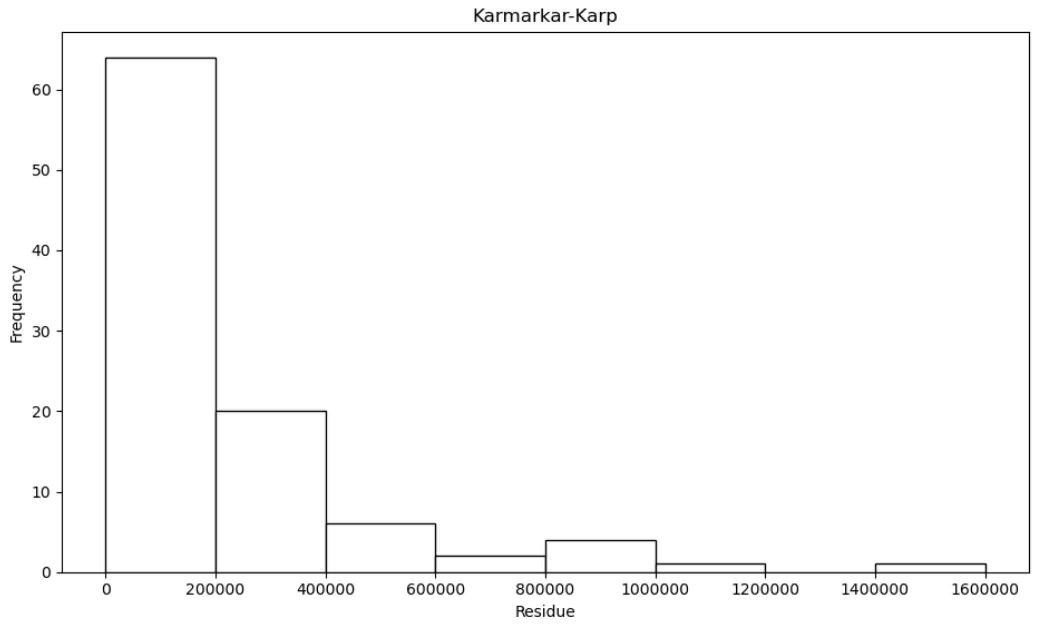
When comparing average residues, we see that each of the algorithms perform considerably worse than Karmarkar-Karp, with the hill climbing algorithm performing slightly better than the random and simulated annealing algorithms without prepartitioning. After prepartitioning, all of the algorithms perform considerably better, due to the fact that Karmarkar-Karp is run on each iteration. The prepartitioned simulated annealing algorithm returns, on average, the lowest residue, followed by the prepartitioned hill climbing algorithm, which is expected because while the hill climbing algorithm might be trapped by a local minimum, the simulated annealing algorithm may be able to escape it by moving to a worse neighbor. Lastly, the prepartitioned random algorithm returns highest average residue of the three algorithms.

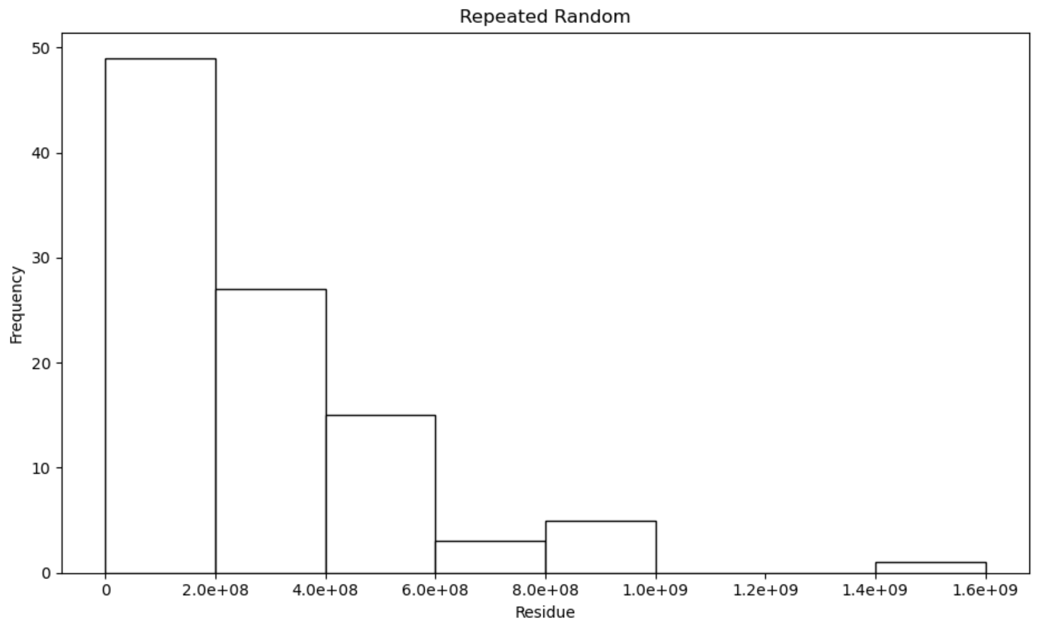
We also created histograms plotting residues and their frequencies for each of the seven algorithms, located in the appendix. Note that all of the histograms are skewed right; therefore, the median residue may be a better metric to diminish the effect of outliers.

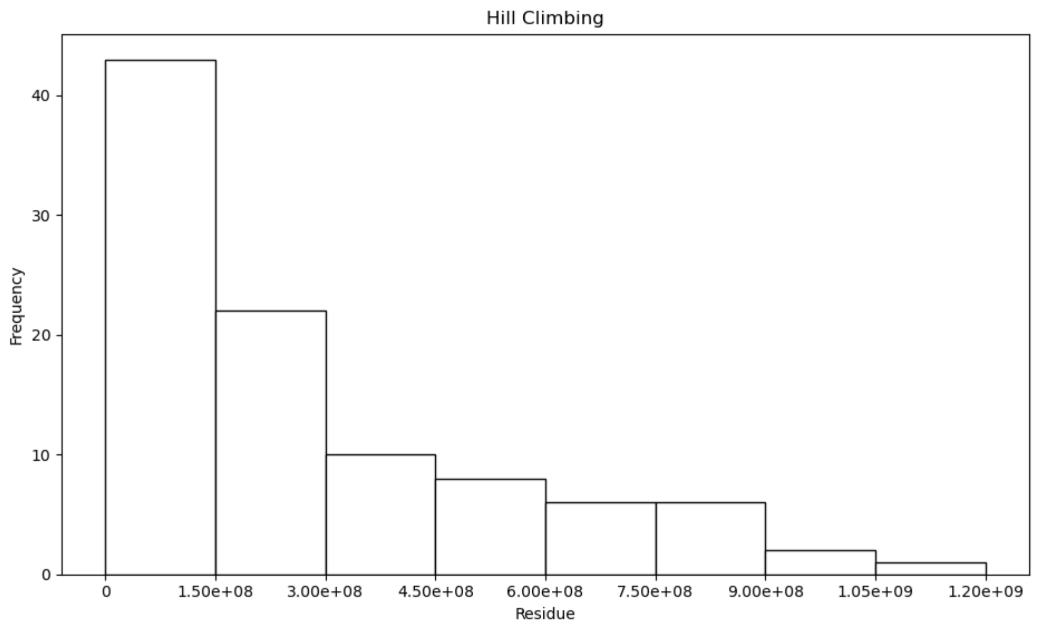
For the three standard algorithms, we could use the Karmarkar-Karp algorithm as an upper bound for desirable residues. For example, given the repeated random algorithm, each iteration could repeatedly generate a random solution until its residue is below that found by Karmarkar-Karp. Then, if the solution is better than the current one, it would replace it as the current best. However, this would increase the runtime because the algorithm would generate more than 25000 solutions in 25000 iterations.

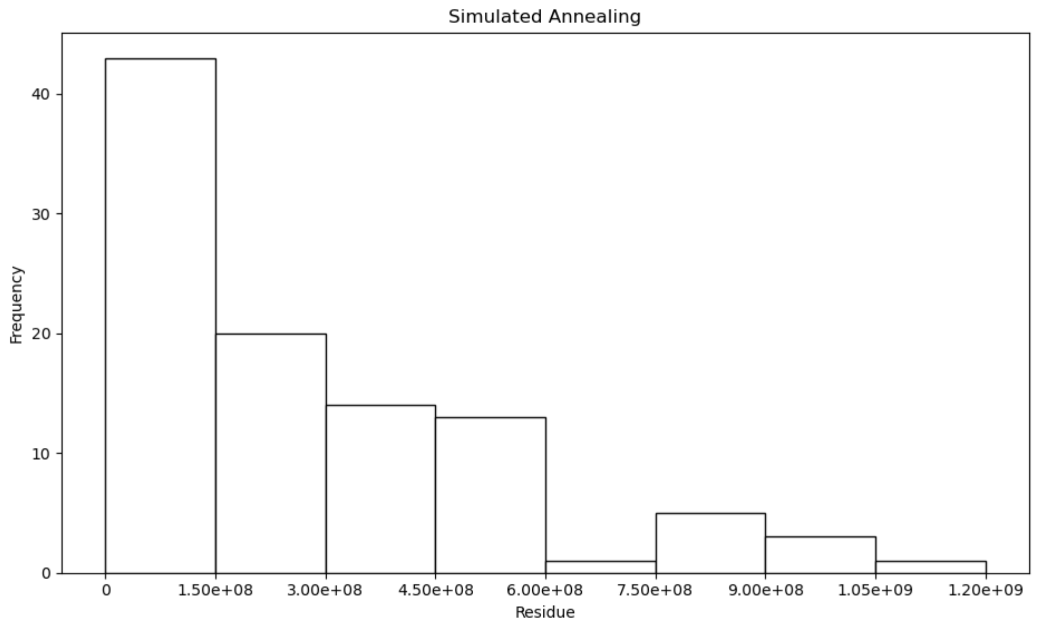
A similar idea can be used to improve the hill-climbing and simulated annealing algorithms. Here, we would find a desirable starting solution with a residue lower than the Karmarkar-Karp residue. Then, we can proceed as normal, moving to neighboring solutions that have a smaller residue (and moving to worse neighbors with some probability for the annealing algorithm). It would take extra time to find a random initial solution, but subsequent iterations would run in the same amount of time as before.

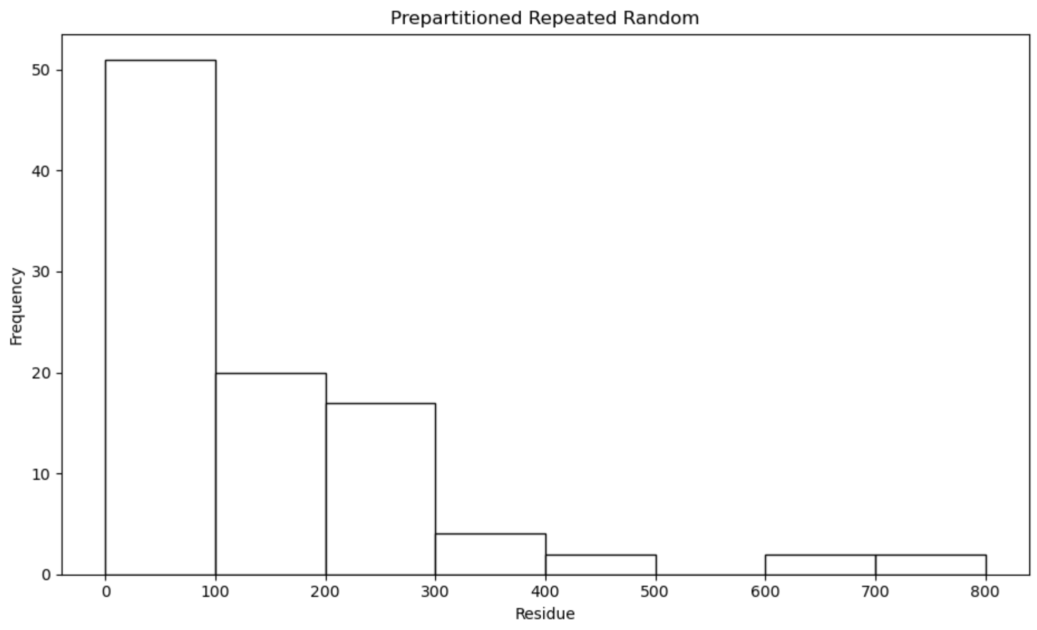
Appendix: Histograms of Results

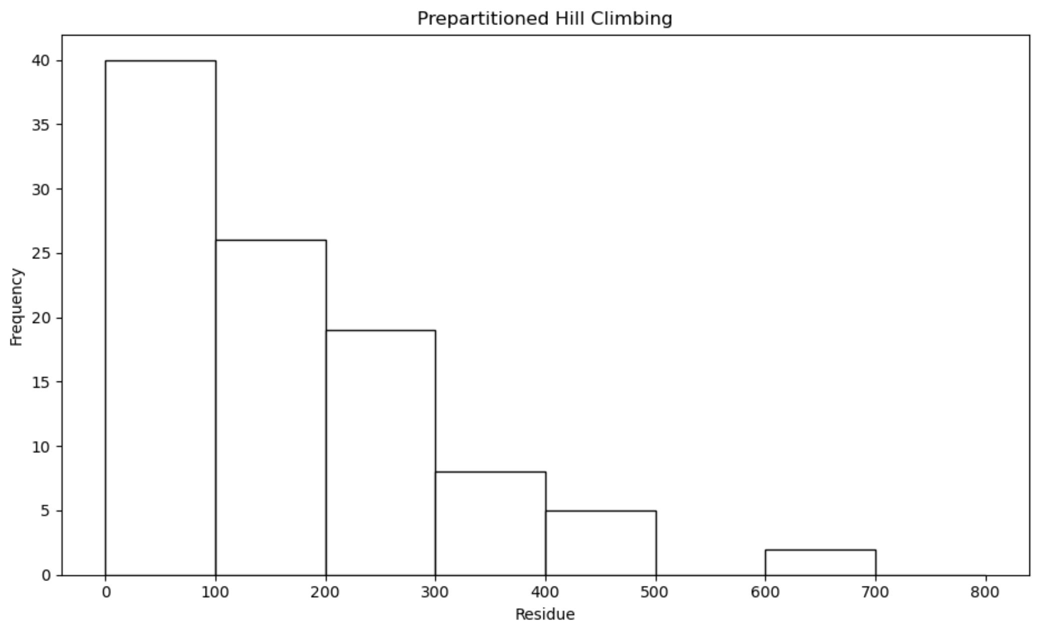


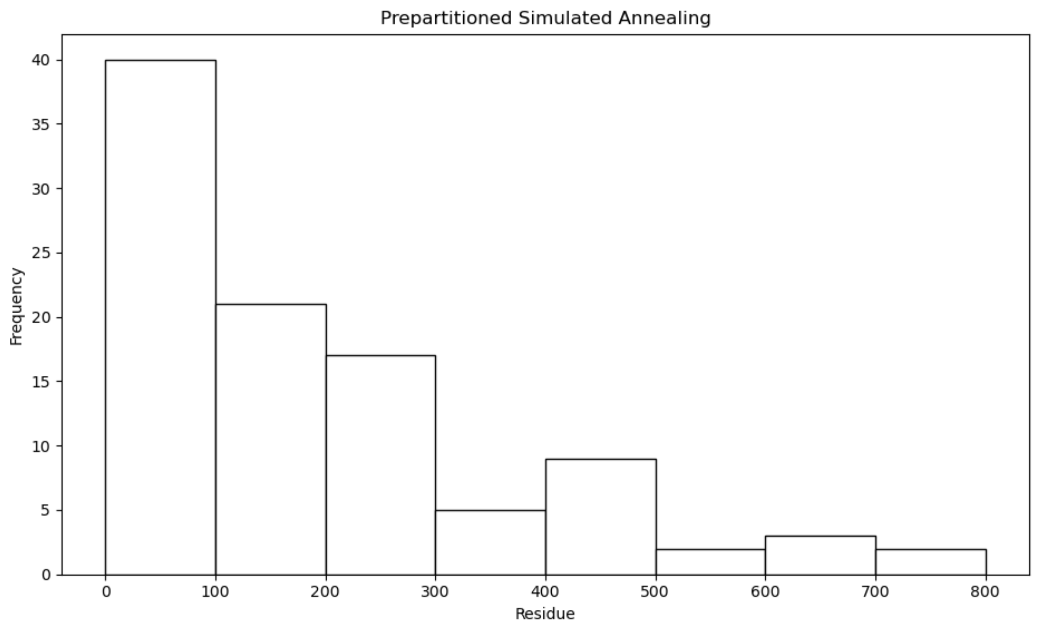



Hill Climbing
Frequency
Residue
0
10
20
30
40
0
1.50e+08
3.00e+08
4.50e+08
6.00e+08
7.50e+08
9.00e+08
1.05e+09
1.20e+09


Simulated Annealing
Frequency
Residue
0
10
20
30
40
0
1.50e+08
3.00e+08
4.50e+08
6.00e+08
7.50e+08
9.00e+08
1.05e+09
1.20e+09


Prepartitioned Repeated Random
Frequency
Residue
0
10
20
30
40
50
0
100
200
300
400
500
600
700
800


Prepartitioned Hill Climbing
Frequency
Residue
0
5
10
15
20
25
30
35
40
0
100
200
300
400
500
600
700
800



Prepartitioned Simulated Annealing
Frequency
Residue
0
5
10
15
20
25
30
35
40
0
100
200
300
400
500
600
700
800