Programming Assignment 3

Jennifer Guo and Ayana Yaegashi

April 20, 2022

1 Introduction

In this report we will discuss the results of our experiments as we examine different solutions to the NP-complete problem, Number Partition. We will compare the results of running Karmarkar-Karp's algorithm as well as three randomized heuristic algorithms: Repeated Random, Hill Climbing, and Simulated Annealing, on 50 instances of the Number Partition problem. We will examine the three algorithms on two different representations of a solution, the standard sequence of signs indicating which group each number belongs in, and a prepartitioned representation indicating which numbers must belong to the same group. Additionally, we will also briefly discuss a dynamic programming pseudo-polynomial solution to Number Partition. We used C++ to implement our algorithms and to run experiments.

2 Dynamic Programming Solution to Number Partition

The Number Partition problem is defined as follows:
Input: a sequence of numbers $A = (a_1, a_2, ..., a_n)$.
Output: partitioning the numbers into two sets denoted by a sequence $S = (s_1, s_2, ..., s_n)$ such that $s_i = \{+1, -1\}$, we output the minimal (optimal) residue of these two sets (the absolute value of the difference of the sums of each set). We define the residue u as

$$u = \left| \sum_{i=1}^{n} s_i a_i \right|$$

Let b be the sum of all numbers in A. Partitioning A into sets 1 and 2, if it is possible to do so such that the residue is exactly 0, then the numbers in both set 1 and set 2 individually sum to $b/2$. If there is no way to partition the numbers such that the residue is exactly 0, one of the sets will have a smaller sum than the other. Without loss of generality, say that the elements in set 1 sum to less than $b/2$, for instance $b/2 - k$ for some integer $0 < k < b/2$. Then, the elements in set 2 sum to greater than $b/2$, i.e. $b/2 + k$. We know that the elements in set 1 must sum to $\leq b/2$ and thus we can construct a dynamic programming algorithm to find the elements in set 1. All other elements must be in set 2.

Observe that the residue is equal to $2k$, where k is the difference between $b/2$ and the sum of set 1 (or equivalently set 2). By maximizing the sum of elements in set 1, up to $b/2$, we minimize k and therefore we minimize the residue of the partition.

Definition: Let $D[i][j]$ be the maximum sum of elements $a_1, ..., a_i$ that is at most j. Thus, since the elements in set 1 add to at most $b/2$, then $D[n][b/2]$ gives the sum of set 1. The elements in set 2 sum to $b - D[n][b/2]$, and the final minimal residue is given by $(b - D[n][b/2]) - (D[n][b/2]) = b - 2D[n][b/2]$.

Base Cases:

$$D[i][0] = 0$$

since the best possible sum of at most 0 with non-negative integers is always 0.

$$D[0][j] = 0$$

since there is no 0^{th} element so the best we can do with at most 0 elements is a sum of 0.

$$D[i][j] = D[i-1][j] \text{ if } j < a_i$$

since it is not possible to sum to at most j using the i^{th} term if the value of the i^{th} term is greater than j.

Recursion:

$$D[i][j] = \max(D[i-1][j], D[i-1][j-a_i] + a_i)$$

There are two ways to get as close to a sum of j as possible with up to i elements. First, we can not include a_i, in which case the best way to get as close to a sum of j with up to i elements is the same as the best way with up to $i-1$ elements. The second way is to include element a_i, in which case we need the best way to get the remainder of the sum $j - a_i$ with up to $i-1$ elements. Taking the maximum of these two sums, we are ensured that we will get as close as possible to a sum of j if not exactly a sum of j.

Analysis: There are $O(n \cdot b/2) = O(nb)$ subproblems since $1 \leq i \leq n$ and $0 \leq j \leq b/2$. To solve each subproblem, we access in constant time the values of previous subproblems and also do one multiplication and one comparison between two values. Thus, solving each subproblem takes $O(nb)$ time so in total the running time is $O(nb)$. Note that the input is $O(n \log b)$ since we receive a sequence of n numbers each represented by at most $\log b$ bits. Thus, the running time is not actually polynomial in the size of the input, although it may look to be so, and thus the running time is "pseudo-polynomial". The space complexity is also $O(nb)$ since there are $O(nb)$ subproblems and each subproblem stores one element in it.

3 Implementation

3.1 Karmarkar-Karp Algorithm

We implemented the Karmarkar-Karp Algorithm using a binary heap, which we implemented ourselves, in order to be able to efficiently query for the largest and second largest elements in A. In order to construct a heap from the input array A, we make n insertions in the heap, each of which takes $O(\log n)$ time. Thus, building the heap takes $O(n \log n)$ time.
Then, each time we take the two largest elements in A, we Delete-Max two times, each of which takes $O(\log n)$ time. Then, if the difference between these two largest elements is greater than 0, we make one insertion in time $O(\log n)$ time. These three operations are performed $O(n)$ times. Observe that each time we take the two maximum elements from A, we add back to the heap at most 1 element. Thus, we can perform these three operations at most $n-1$ times before there is only one non-zero element in the heap. Thus, in total this part of the algorithm takes $O(n \cdot 3 \log n) = O(n \log n)$ time. As a whole, the algorithm therefore takes $O(n \log n)$ time.

3.2 Heuristics

We wrote functions to implement the three heuristics, Repeated Random, Hill Climbing, and Simulated Annealing for both the sequence representation and the prepartitioned representation. We first

wrote functions that would calculate the residue given an instance A of the partition problem and a solution S for both the sequence and prepartition representations. For the prepartitioned representation, we first had to force the instance A to agree with the solution S by turning it into the sequence A' that enforced the prepartioning from S. We did so by implementing the pseudocode from the programming assignment description, summing all elements together that were in the same prepartitioned group. Then we were able to run Karmarkarp-Karp on A' to obtain the residue. Finding the residue for the sequence representation solution was more straightforward, as we simply had to find the difference between the sums of the two groups, as represented by the signs.

Then, for both the sequence and prepartitioned representations, we wrote functions that would generate a random solution as well as generate a random neighbor to a previous solution S. Using these helper functions, we were able to implement the Repeated Random, Hill Climbing, and Simulated Annealing algorithms using the pseudocode given to us in the programming assignment write-up.

4 Results

	Mean	Median	Std. Dev.	Min	Max
Karmarkar-Karp	245386.8	121184.5	354371.5	10499.0	2056252.0
Sequence					
Residue (start)	3958729601726.5	3130627979407.5	2996291823343.0	66442474389.0	14254129185782.0
Repeated Random	258316086.5	166296677.0	249409125.6	23256623.0	976991974.0
Hill Climbing	322931071.4	258625287.5	320630546.9	6217284.0	1518342799.0
Sim. Annealing	276765850.9	212548760.0	194090398.3	332443.0	916255580.0
Prepartitioning					
Residue (start)	6599513.6	3427150.5	7618893.7	34814.0	34056144.0
Repeated Random	132.6	109.0	103.0	2.0	418.0
Hill Climbing	712.7	417.0	826.1	10.0	3940.0
Sim. Annealing	231.4	156.0	234.3	3.0	786.0

Table 1: Our results over 50 instances of the partition problem, using 25,000 iterations of each heuristic algorithm. Note that the row titled "Residue (start)" refers to the residue of the random instance before any heuristic was applied. Thus, this is the starting residue we were trying to improve upon by implementing various heuristics.

Algorithm	Average Runtime Per Trial (ms)
Karmarkar-Karp	0.049387
Sequence Repeated Random	129.48
Sequence Hill Climbing	14.059
Sequence Simulated Annealing	15.122
Prepartitioning Repeated Random	405.23
Prepartitioning Hill Climbing	223.18
Prepartitioning Simulated Annealing	235.63

Table 2: Average runtime per each trial in milliseconds for each of the algorithms.

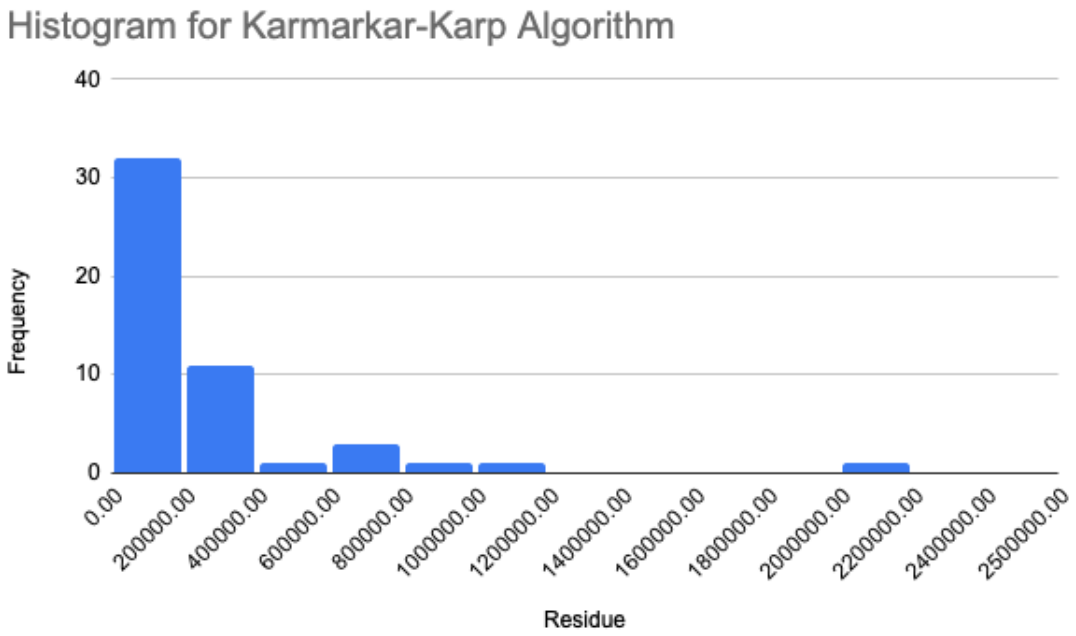

Figure 1: Histogram of the frequency of different residues for the Karmarkar-Karp algorithm over 50 trials.

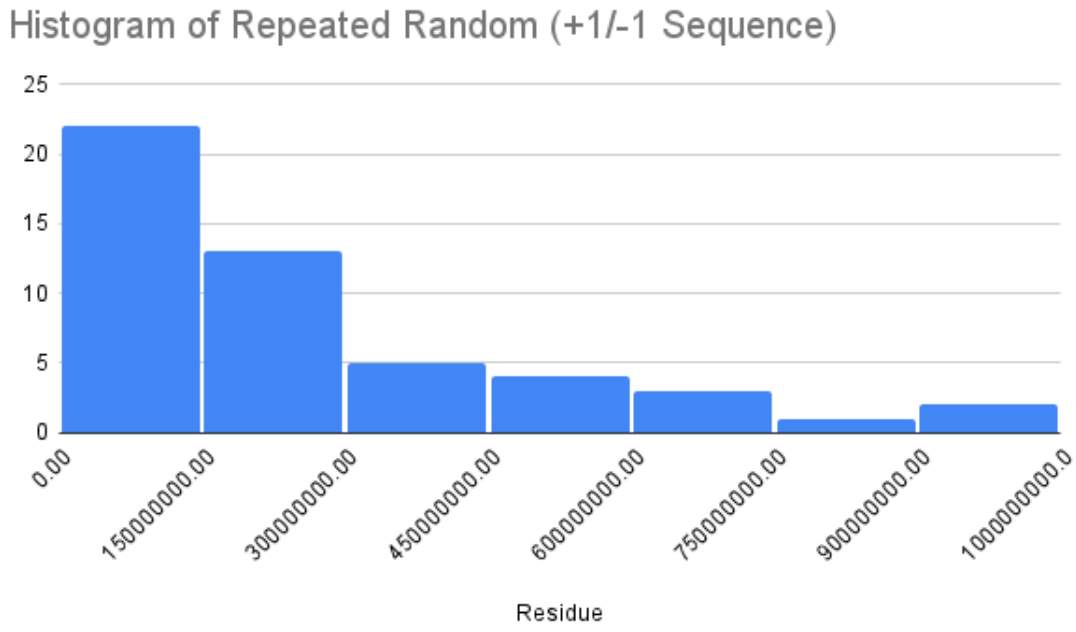

Figure 2: Histogram of the frequency of different residues for the Repeated Random algorithm for the sequence representation over 50 trials.

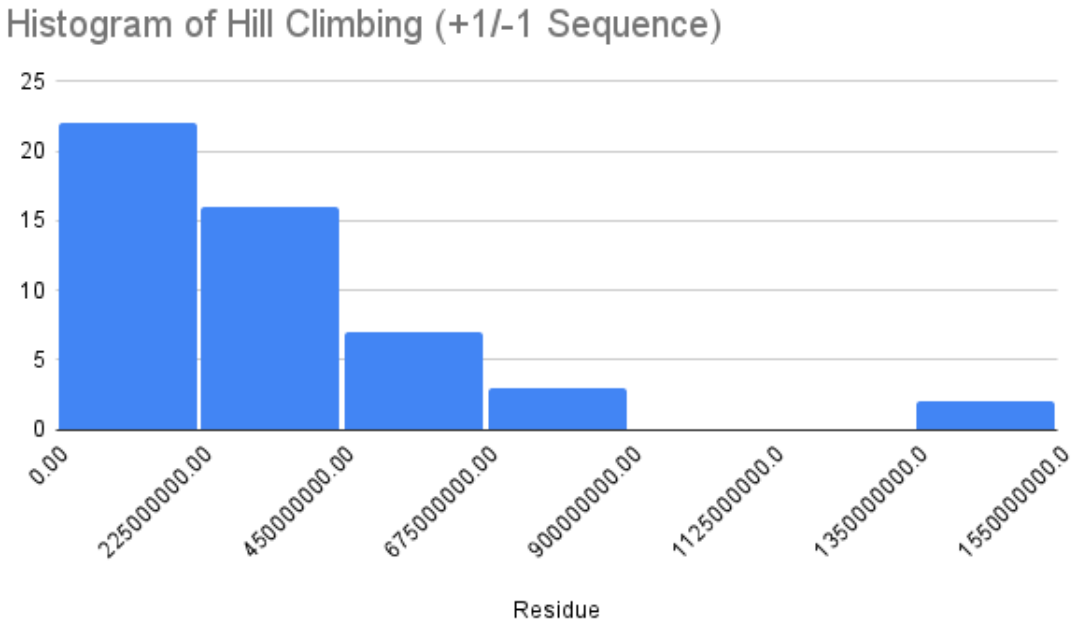

Figure 3: Histogram of the frequency of different residues for the Hill Climbing algorithm for the sequence representation over 50 trials.

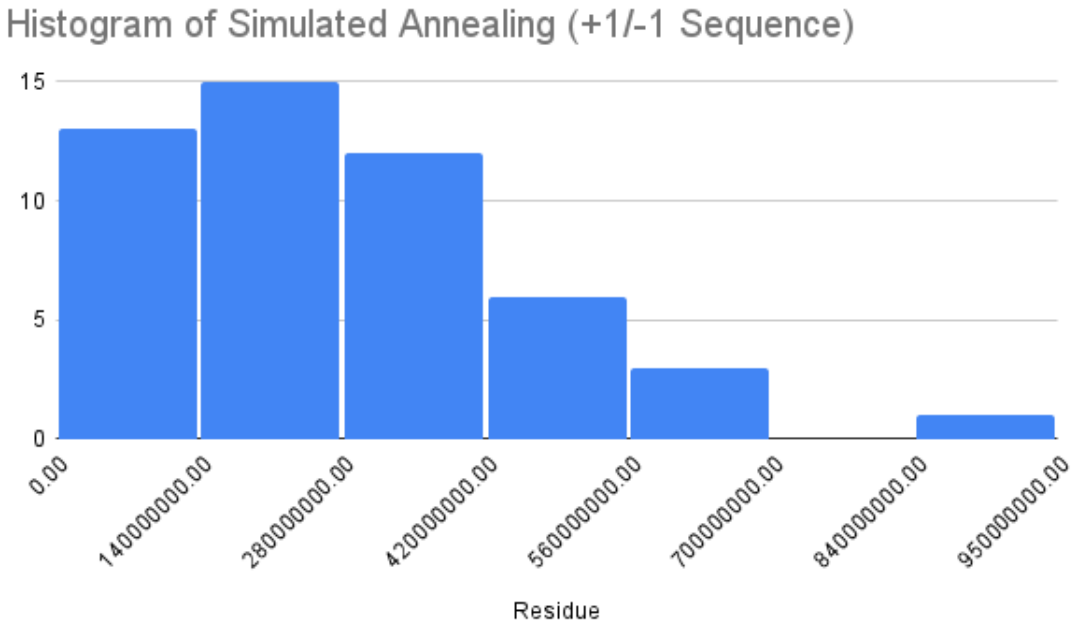

Figure 4: Histogram of the frequency of different residues for the Simulated Annealing algorithm for the sequence representation over 50 trials.

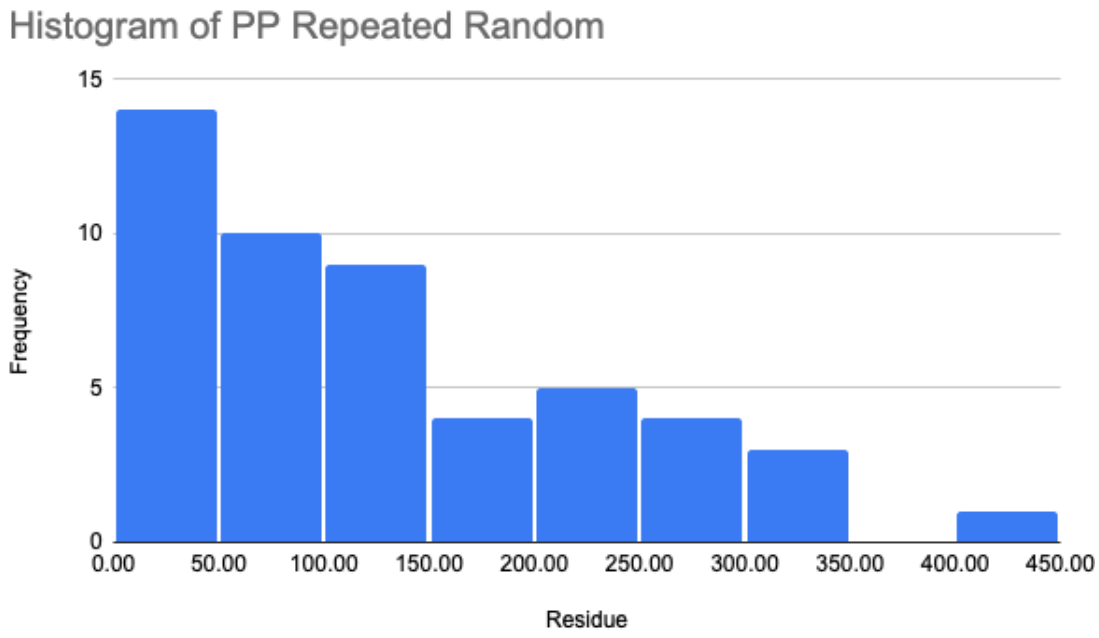

Figure 5: Histogram of the frequency of different residues for the Repeated Random algorithm for the prepartitioned representation over 50 trials.

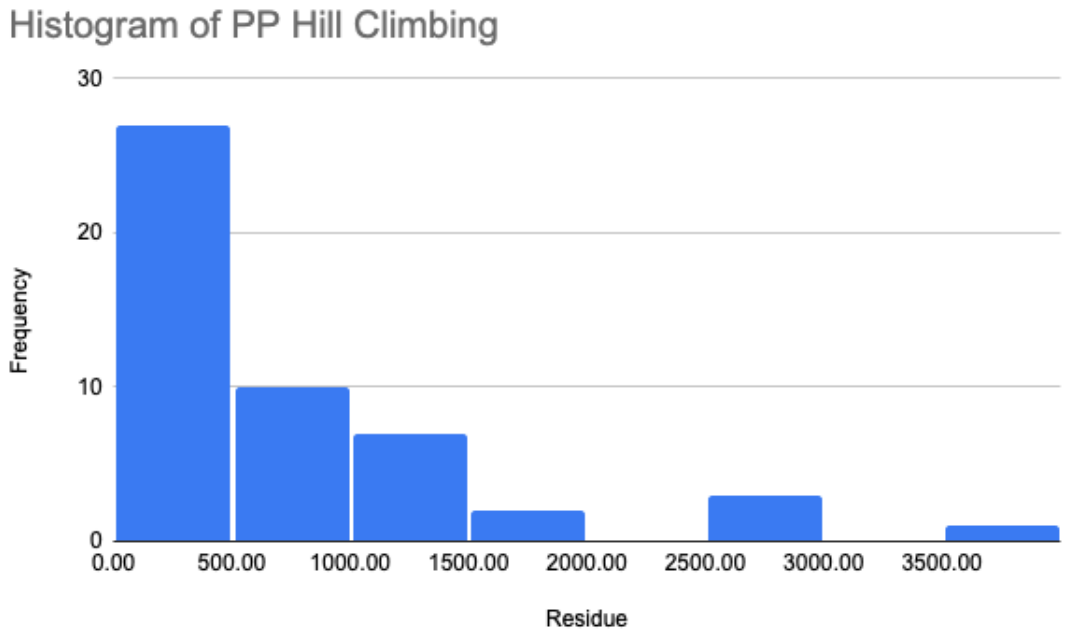

Figure 6: Histogram of the frequency of different residues for the Hill Climbing algorithm for the prepartitioned representation over 50 trials.

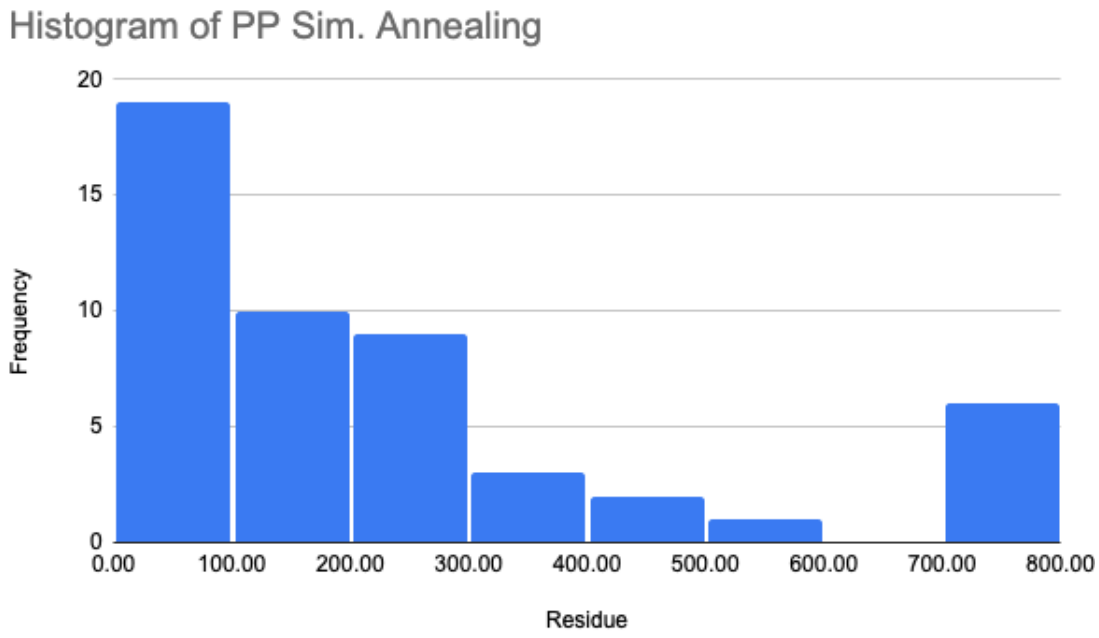

Figure 7: Histogram of the frequency of different residues for the Simulated Annealing algorithm for the prepartitioned representation over 50 trials.

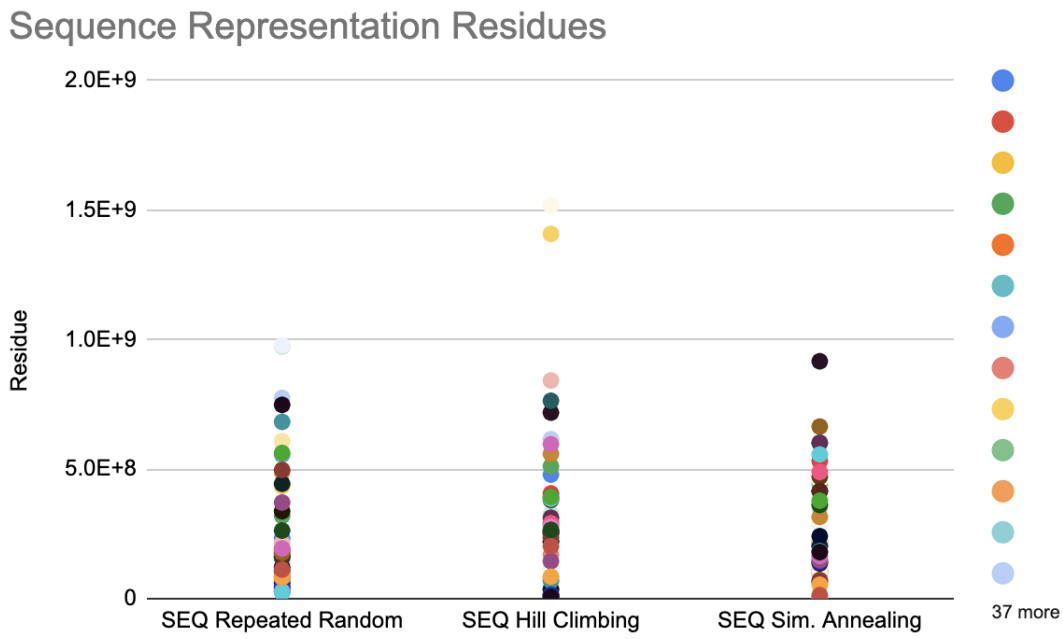

Figure 8: Scatterplot comparison of the residues over 50 trials for each of three heuristic algorithms for the sequence representation.

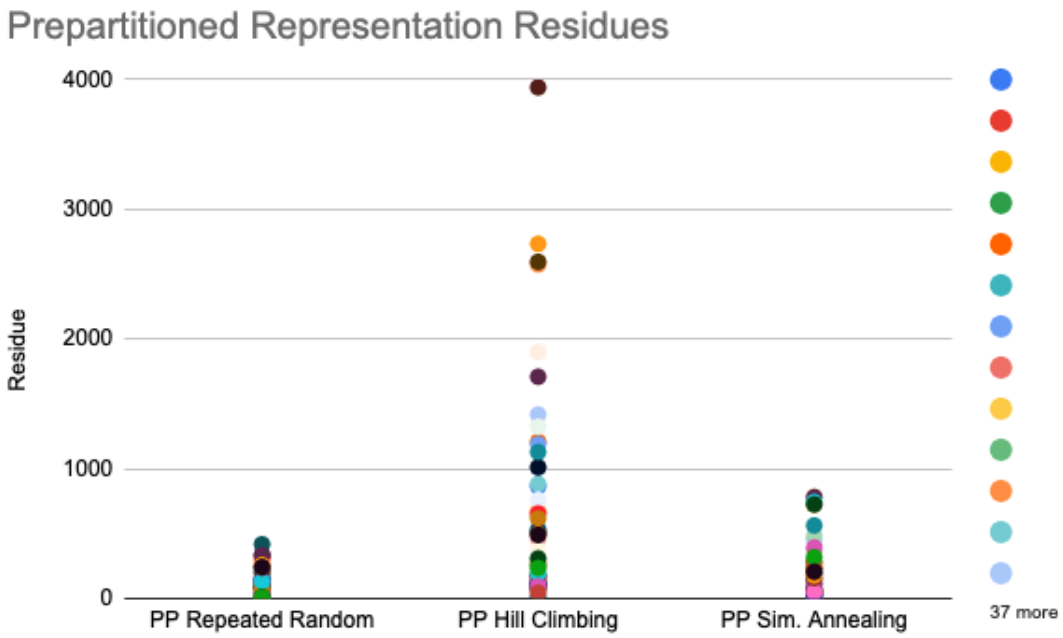

Figure 9: Scatterplot comparison of the residues over 50 trials for each of three heuristic algorithms for the prepartitioned representation.

5 Discussion

5.1 Results

It is clear from Table 1 that the prepartitioning representation tends to produce a better solution with a lower residue than the sequence representation. On average, using the prepartitioning representation produces solutions with residues a million times (a factor of 10^6) better than the sequence representation. Compared to the Karmarkar-Karp algorithm, the prepartitioning representation produces solutions that are better by a factor of 10^3, while the sequence representation produces solutions that are worse by around a factor of 10^3. Clearly, the prepartitioning representation is better than the sequence representation for finding a better solution. This could be attributed to the fact that solutions in the sequence representation assigns every number to a set, while solutions in the prepartitioning representation only restricts a few numbers to be grouped together. Therefore, the prepartitioning representation offers greater flexibility within a solution. In particular, we apply the Karmarker-Karp algorithm to the prepartitioning, which explicitly tries to minimize the residue. On the other hand, because the sequence representation randomly assigns elements to different sets, all heuristics rely on randomly getting to a better assignment without applying any sort of algorithmic logic (such as the Karmarkar-Karp algorithm). Especially if we begin with an unfavorable split for the sequence representation and we are left to search within this split's random neighbors (as in the Hill-Climbing and Simulated Annealing heuristics), the optimal solution may be far away in terms of neighboring solutions. Therefore, using the prepartitioning representation for the number partition problem certainly produces more favorable results, however, at the cost of runtime, which we found can be much higher for the prepartitioning representation than the sequence representation (as seen in Table 2). A more in-depth runtime discussion will be in the next subsection.

For both the sequence and prepartitioning representations, it appears that on average, the repeated random algorithm does the best and the hill climbing algorithm does the worst, with simulated an-

nealing somewhere in between. All 3 algorithms improve on the initial randomized solution by a factor of 10^4, however. Both repeated random and hill climbing have tradeoffs. For the repeated random approach, if we come upon an optimal solution, we may not be able to continue exploring around this optimal solution and making incremental improvements. On the other hand, while the hill climbing approach allows for exploration around optimal solutions, we may get stuck at a local optimum. Theoretically, it makes sense that simulated annealing would do better than hill climbing, but not as good as repeated random because simulated annealing combines the randomness of potentially going to a worse neighbor (in order to be able to eventually reach a better optimum overall) without venturing too far from a known optimum. The hill climbing algorithm likely does worse overall because it depends heavily on the initial random solution being a good one. Otherwise, hill climbing can easily get stuck at a local optimum solution without finding a global optimum. The repeated random algorithm likely does better overall since it can look at many solutions without ever getting stuck. Furthermore, since we iterate $25,000$ times for each algorithm, repeated random can explore $25,000$ different solutions each with varying residues all over the solution map, likely attributing to it doing the best overall. We might expect that had we done fewer iterations, the results for the repeated random approach may not have been as good.

In the histograms, we noticed that for both Hill Climbing using both the sequence and prepartitioning representations, there were a couple of instances whose residues were very high. For the sequence representation, there were 2 instances whose residues were between 1,350,000,000 and 1,550,000,000. For prepartitioning, there were 3 instances with residues between 2,500 and 3,000, and one with a residue between 3,500 and 4,000. Since the repeated random for these same instances has no residues above 450, we believe these were instances where Hill Climbing got stuck at local minima and thus were not able to reach the global minima.

You will notice that the maximum residues produced by Simulated Annealing are lower than the maximum residues from Hill Climbing. In particular, comparing the histograms for Simulated Annealing and Hill Climbing we can see evidence that because Simulated Annealing allows for movement to "worse" solutions, this enabled the algorithm to eventually reach better local minima than in the Hill Climbing approach.

5.2 Experimental Runtimes

We noticed several trends in our experimental runtime data. First, we found that the experimental runtimes for the sequence representation were faster than those for the prepartitioning representation. We believe this makes sense, since using the prepartitioning representation requires first transforming the input A into a restricted input A' as well as running the Karmarkar-Karp algorithm on each instance. On the other hand, using the sequence representation we can directly calculate the residue by adding up up the sums of each set and taking the difference. This is $O(n)$ addition/subtraction operations. On the other hand, as discussed in section 3.1, just running the Karmarkar-Karp algorithm alone requires $O(n \log n)$ time as implemented with a binary heap (which was our implementation). Thus, we believe it makes sense that experiments using the sequence representation were faster than those using the prepartitioning representation, since this is reflected in the theoretical running times as well.

We also noticed that within each representation, the Repeated Random heuristic had a longer experimental running time than either the Hill Climbing or Simulated Annealing running times. We believe this is because for the Repeated Random heuristic, each of the 25,000 iterations requires constructing an entirely new sequence or prepartitioning of $O(n)$ numbers. On the other hand, for Hill Climbing and Simulated Annealing, we change at most 2 elements in the existing representation to get to a neighbor. In other words, we make $O(1)$ modifications to the existing representation.

The experimental runtime data introduces some important tradeoffs between the different heuristics. As discussed above, we found that in general using the prepartitioning representation led to better overall residues. However, from a practical standpoint we can see in the runtime data that using the prepartitioning representation is also slower. A similar tradeoff exists among heuristics. While

the Repeated Random heuristic tended to yield better (smaller) residues, this heuristic also had slower experimental runtimes. This being said, between the Hill Climbing and Simulated Annealing heuristics, we found a negligible difference between experimental runtimes. However, in the residue data we collected, Simulated Annealing yielded better residues. Thus, between the two heuristics, it would appear that for a very small sacrifice in running time, we can get better residues by using the Simulated Annealing heuristic instead of the Hill Climbing heuristic.

5.3 Using Karmarkar-Karp as a Starting Point

If we were to find the solution given by Karmarkar-Karp and use it as the starting point for the randomized heuristic algorithms, we believe our results would generally improve. First, we would want to augment our implementation of the Karmarkar-Karp algorithm to return not just an approximate, minimal residue of the sequence, but also the partition of numbers that led to this residue. We can then use this partition returned by Karmarkar-Karp as a starting point. Using the repeated random approach would not take advantage of this starting point, since the subsequent partitions it explores would be unrelated to our starting point. However, we believe using the Hill Climbing and Simulated Annealing approaches could help get better approximations. We could define neighbors similarly to how we defined neighbors for the sequence representation, namely with one or two elements in different sets. The Hill Climbing approach would allow us to get to a better optimum if the Karmarkar-Karp starting point is not already a local optimum. However, it would run the risk of getting stuck at a local optimum and never reaching a global optimum. On the other hand, using the Simulated Annealing approach would give us the best chance of not just finding the nearest local optimum, but also potentially finding the global optimum since there is a chance that we move to a "worse" solution using simulated annealing. Because the Hill Climbing and Simulated Annealing for the sequence implementation is currently returning solutions that are worse than the approximated solution from Karmarkar-Karp, if we simply used the Karmarkar-Karp solution as a starting point, as a baseline the resulting solutions for the hill climbing and simulated annealing algorithms would be at least as good as the Karmarkar-Karp solution, improving the solution by a factor of 10^3 for the sequence representation. Thus, we believe that using Karmarkar-Karp as a starting point will yield better results than any of the results for the sequence representation.