Heuristic Algorithms for Solving the Number Partition Problem

Thomas N. Brooks and Eric Y. Li

April 2020

Abstract

We test the performance of several heuristic algorithms to minimize the residue of splitting a set of numbers into two disjoint subsets. We consider inefficient dynamic programming solutions to this problem as well.

1 A Dynamic Programming Solution to the Number Partition Problem

We are going to present a dynamic programming solution to the number partition problem that runs in exponential time. Suppose that we have an array A of integers. Let $D(i, j)$ equal *true* if there exists a subarray of $A[1, \cdots, i]$ with sum j and equal *false* otherwise. Recursively,

$$D(i,j) = \begin{cases} \textit{true}, & j = 0 \text{ or } D(i-1,j) \text{ or } D(i-1, j-A[i]) \\ \textit{false}, & \text{otherwise} \end{cases}$$

First, we show the correctness of the recursion. Suppose $j = 0$. Then the empty subarray sums to zero, so regardless of i, D evaluates to true. Now let $j > 0$. Assume that $i > 0$. Otherwise, D trivially evaluates to false because the sum of elements of an empty array cannot exceed zero. Then if there exists a subarray end-indexed at $i-1$ whose sum is j, that same subarray is by definition end-indexed at i. So $D(i, j) = \textit{true}$. In addition, if there exists an $i-1$-end-indexed subarray whose sum is precisely $A[i]$ less than j, then we can construct an i-end-indexed subarray containing precisely those elements in the former subarray plus $A[i]$ whose sum is j. By exhaustion, otherwise, $D(i, j) = \textit{false}$.

We claim that the solution to the number partition problem is

$$\min_{i \in [0, \cdots, \lfloor \frac{\sum_j A[j]}{2} \rfloor]} \left\{ \left(\sum_j A[j] \right) - 2i \;\middle|\; D(|A|, i) = \textit{true} \right\}$$

We know that finding the relevant i will give us the optimally minimal residue, since $D(|A|, i) = \textit{true}$ guarantees that there exists a subset of A whose sum is i,

and minimizing $\left(\sum_j A[j]\right) - 2i$ minimizes the difference in size of the partition consisting of the subarray whose sum is i and its complement.

This algorithm must iteratively compute $D(i, j)$ for $i \in \{1, \cdots, |A|\}$ and $j \in \{1, \cdots, \lfloor \frac{\sum_j A[j]}{2} \rfloor\}$, so the result is an algorithm that runs in time $O(|A|b)$, where the largest number in A is of $\log b$ bits.

2 An Efficient Implementation of the Karmarkar-Karp Algorithm

The Karmarkar-Karp algorithm is a deterministic heuristic for the number partition problem. Given an array of numbers, we repeatedly take the two largest numbers in the array and replace one with their difference, the other with zero. In doing so, we essentially assign these numbers to opposite sides of the partition, so we can just consider the residue they contribute. To implement this algorithm naively, we could find the maximums by walking through the array, replace the values in the array in linear time, and do this until there is only one element left. In each iteration, we do a linear amount of work, and in each iteration we remove two maximum values and replace with one value (not counting zero), so the number of values to consider decreases by 1. To have only one value remaining, we would have had to do this $n - 1$ times, where n is the length of the array. Thus, we do $O(n)$ work for $O(n)$ iterations, resulting in an $O(n^2)$ naive algorithm.

However, since we need to find the maximum of the array many times in this algorithm, it lends itself well to using a max-heap. If we put all of the values in the array into a heap, we just need to execute two $extractMax$ operations (to get the two largest values) and one $insert$ operation (to insert their difference) per iteration, each of which are $O(\log n)$ using a binary heap representation. There are still $O(n)$ iterations, but this means we have now improved to an $O(n \log n)$ algorithm, which is the implementation we use here.

The solution is as follows:

Algorithm 1: Karmarkar-Karp algorithm with max-heap

```
Data: Max-Heap H containing the values in the input array.
Result: Residue returned by Karmarkar-Karp algorithm.
while size(H) > 1 do
    largest = extractMax(H)
    second = extractMax(H)
    insert(H, largest-second)
end
return extractMax(H)
```

	KK	RR	RR (PP)	HC	HC (PP)	SA	SA (PP)
1	945859	1.56e8	43	9.25e7	179	9.80e8	295
2	41781	1.53e9	415	9.26e8	43	5.66e8	13
3	5492	3.21e7	234	4.60e7	66	1.36e8	378
4	17300	4.37e7	306	3.40e8	920	2.98e8	48
5	25437	7.30e8	129	1.44e8	339	1.50e8	65
6	30773	8.17e7	363	1.14e7	79	7.49e7	63
7	124782	5.52e8	24	4.18e7	50	7.09e8	182
8	16982	1.03e8	18	4.68e8	6	6.18e7	22
9	512219	2.26e8	117	5.83e7	149	4.10e8	31
10	733810	1.76e8	24	2.20e8	0	3.01e8	396
11	113418	8.84e7	12	179e8	254	2.61e8	152
12	235236	1.99e8	90	7.81e8	50	1.53e8	292
13	122233	1.18e8	337	5.22e7	1	2.68e8	165
14	6778	2.70e8	200	3.95e8	26	1.41e8	38
15	1084361	1.04e8	29	2.74e8	225	2.17e8	473
16	429527	1.82e8	427	2.03e8	133	5.20e7	159
17	40220	5.70e8	76	3.12e8	172	8.94e6	74
18	12523	8.53e6	135	3.91e8	67	7.60e7	31
19	195626	3.31e8	70	4.93e8	342	6.52e7	498
20	10848	1.15e8	168	8.01e7	220	2.37e8	108
Means	254015	3.63e8	167	2.34e8	183	3.69e8	149

Table 1: Residues from twenty representative trials of each of the heuristic algorithms for solving the optimal number partition problem. The bottom row shows statistical mean over all 100 trials.

3 Experiment, Data, and Analysis

We conducted 100 trials of the following experiment. We generated a random array of longs in the range $[0, 10^{12}]$, and used each heuristic algorithm (Karmarkar-Karp, repeated random, repeated random with prepartition, hill climb, hill climb with prepartition, simulated annealing, and simulated annealing with prepartition) to find a near-optimal (minimal) residue. We present in the tables below data from a representative set of twenty of these 100 trials:

	KK	RR	RR (PP)	HC	HC (PP)	SA	SA (PP)
1	0	33	811	3	817	8	1896
2	0	32	812	3	719	8	1927
3	0	33	809	3	728	8	1965
4	0	32	804	3	727	8	1926
5	0	32	859	3	744	9	1999
6	0	32	814	3	722	9	1926
7	0	32	790	3	720	8	1977
8	0	32	821	3	729	8	1989
9	0	32	807	3	716	8	1987
10	0	32	817	3	736	8	1961
11	0	34	1051	3	821	8	2207
12	0	31	811	3	712	8	2069
13	0	30	769	3	698	8	2084
14	0	30	770	3	702	8	1857
15	0	31	767	3	691	8	1864
16	0	31	773	3	697	8	1839
17	0	30	770	3	707	8	1882
18	0	30	764	3	705	8	1883
19	0	30	775	3	698	8	1855
20	0	31	769	3	698	8	1859
Means	0	32	810	3	755	8	1936

Table 2: Times in milliseconds from twenty representative trials of each of the heuristic algorithms for solving the optimal number partition problem. The bottom row shows statistical mean over all 100 trials.

We can see that in general, the prepartitioning does wonders for the performances of the various heuristic algorithms in terms of optimal residue, although this came at the expense of runtime in every case on account of the repeated need to call Karmarkar-Karp as a subroutine. We expect that the prepartitioning might improve the discovered residue, as the non-prepartitioned algorithms generally search throughout a larger state space than the prepartitioned algorithms, so it is less likely that any given iteration finds a better solution than the original random solution. Meanwhile, conditioned on the fact that each algorithm finds a better solution on a given iteration, the non-prepartitioned algorithms are likely to find a better solution than their prepartitioned counterparts, as the former are drawing conditionally from the uniform distribution on all better solutions, whereas the latter are drawing only from the set of better neighbor solutions.

Of course, in terms of runtime, Karmarkar-Karp takes the cake on account of on the one hand its being deterministic and on the other hand its not requiring the 25,000 iterations used in the other algorithms. All that is require of Karmarkar-Karp in our experiment is to performed a small, fixed number of heap operations which we know to be fast. However, this comes at the expense of residual performance as compared to the prepartitioned algorithms. The reasons discussed above regarding the performance of the non-prepartitioned algorithms account for the consistently better performance of Karmarkar-Karp as compared to the former.