CS124 PAS 3

Tom Shlomi

April 2021

1 Introduction

The Number Partition problem is the problem of, given a set of non-negative integers, splitting the set into two subsets such that the sums of each of those sets are as close as possible to each other. Number Partition is NP-complete, and so not solvable in polynomial time unless P=NP.

2 Pseudo-Polynomial Dynamic Programming Algorithm for Number Partition

Despite being NP-complete, it can be solved in pseudo-polynomial time. That is, it can be solved in $O(nb)$ time, where n is the number of integers in the set and b is the sum of the set.

Consider the following algorithm. Note that the Number Partition problem on a set A is equivalent to finding the subset of A with sum as close to $b/2$ as possible. Initialize, D, an boolean $\lfloor b/2 \rfloor$ by n array. $D[i,j]$th entry of the array should represent whether a set with sum i exists containing only the first j elements of A. If such a set exists, then the jth element of A is either in it or isn't. If it is, then there must be a subset with sum $i - A[j]$ containing only the first $j-1$ elements of A. If it isn't, then there must be a subset with sum i containing only the first $j-1$ elements of A. Thus $D[i,j] = D[i - A[j], j-1] \vee D[i, j-1]$, except when i or j are 0. Since a set with no elements has sum 0, $D[i,0] = D[0,j] = TRUE$ for all i, j. We can thus solve Number Partition using a dynamic algorithm that calculates the elements of D, tracking as the algorithm iterates the element of D which is $TRUE$ that has the highest i. Since $i \leq \lfloor b/2 \rfloor$, the highest i will correspond to a partition that is closest to even. The residue is b minus twice the largest i. Since each element of D requires constant time to calculate, and since D has $O(nb)$ elements, the algorithm takes $O(nb)$ space and time.

3 Karmarkar-Karp

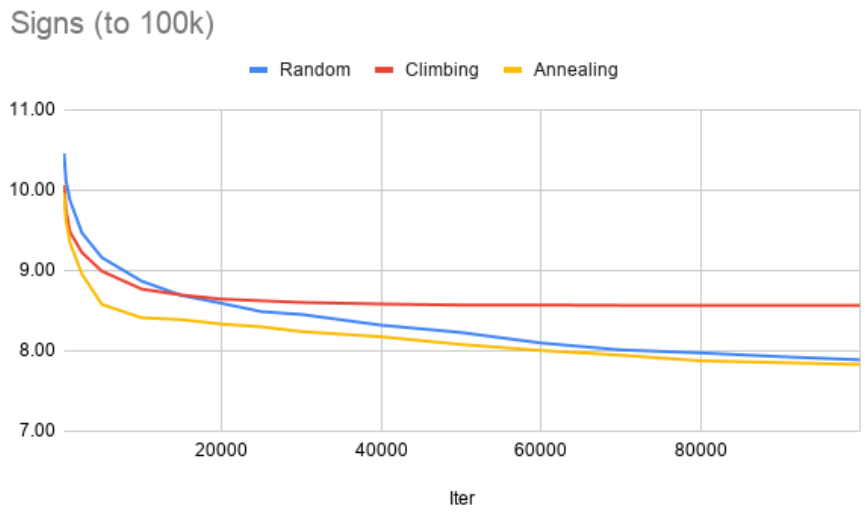
The Karmarkar-Karp provides an approximate solution to Number Partition. If, in a partition, two numbers are put in separate sets, the same residue is achieved by replacing the larger number by their difference and deleting the smaller number. Karmarkar-Karp, at each time step, deletes the top largest numbers from the set and replaces them by their difference. This guarantees returning an attainable residue.

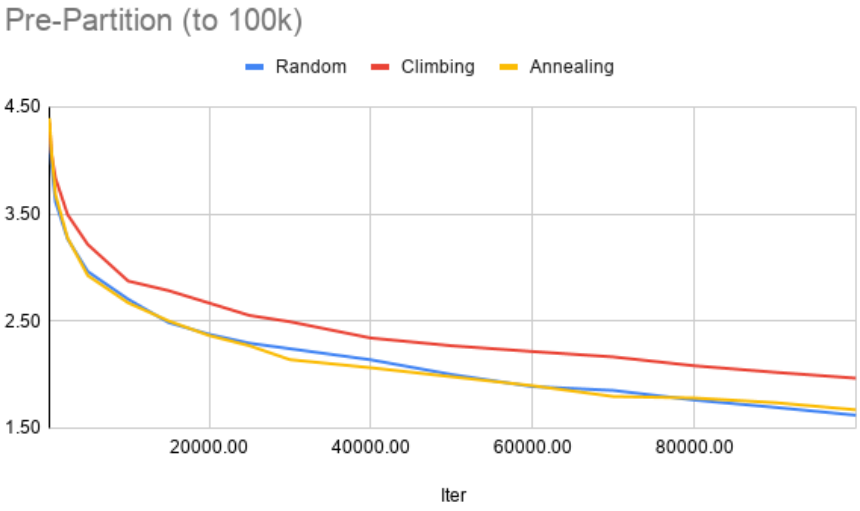
To run Karmarkar-Karp, first sort A. Until A contains 1 element, delete the first two elements of A and insert their difference back into A such that it remains sorted. Return the last element in A. Sorting A takes $O(n \log n)$ time, and inserting an element into the sorted list takes at most $\log n$ time, since A will have at most n elements. Since $n-1$ elements are inserted, Karmarkar-Karp runs in time $O(n \log n + (n-1) \log n) = O(n \log n)$ time.

4 Results

4.1 Residue As A Function of Iter

First of all, we graphed the mean residue for each algorithm and for each form, up to a max iter of 100 thousand. On the x-axis is the iter at which we are graphing, and on the y-axis is the log of the residue.





We can see that for the both the sign method and the pre-partitioning method (in the long-term), climbing did by far the worst. This makes a lot of sense because we would except climbing methods to on average plateau much faster than annealing or random; in fact, because as iter increases our T(iter) function increases, increasing the probability that we switch, we would expect summations to be the only ones that will get completely stuck on a non-optimal after some number of iterations.
We can see this affect more prominently in the signs graph. At the start, climbing actually does much better than random. But there's a crossover at $iter = 15000$, and while climbing plateaus random continues to get better, almost linearly in the log. Annealing, like random, seems to not really plateau, although one can see that the two are getting closer and closer, so we might expect that after some time random will become better than annealing. A similar-ish affect can be seen in the pre-partitioned graph, except the plateaus are all much less significant. This is because there are 50 times more possibilities for each pre-partition group than for each sign, so it will take much longer to for summation and annealing to start getting suck.

4.1.1 Specific Iters

Below we have more in-depth data at iter = 25k, and iter = 100k. in-depth data includes the residue of the $25th$, 50th, and 75th percentiles, the mean, and min and max.
We also put the run of the Karmarker-Karp algorithim by itself in both the 25k and 100k tables so they can be more easily compared. The

At 25k Max_Iter	Karmarkar	Random (Signs)	Climbing (Signs)	Annealing (Signs)
Mean	3.40E+05	3.08E+08	4.20E+08	1.99E+08
Min	7.33E+02	7.84E+05	1.15E+06	3.04E+06
Q1	3.97E+04	8.46E+07	1.42E+08	5.21E+07
Median	9.89E+04	1.94E+08	3.36E+08	1.16E+08
Q3	3.08E+05	4.42E+08	5.66E+08	2.65E+08
Max	1.00E+07	1.51E+09	2.53E+09	9.12E+08

At 25k Max_Iter	Random (Partition)	Climbing (Partition)	Annealing (Partition)
Mean	164	348	207
Min	0	4	0
Q1	44	83	48
Median	118	236	118
Q3	208	460	287
Max	870	1769	972

At 100k Max_Iter	Karmarkar	Random (Signs)	Climbing (Signs)	Annealing (Signs)
Mean	3.40E+05	7.74E+07	3.68E+08	6.77E+07
Min	7.33E+02	7.84E+05	9.63E+05	1.09E+06
Q1	3.97E+04	2.76E+07	1.19E+08	2.37E+07
Median	9.89E+04	5.38E+07	2.57E+08	4.58E+07
Q3	3.08E+05	1.11E+08	5.14E+08	8.58E+07
Max	1.00E+07	4.91E+08	2.53E+09	3.48E+08

At 100k Max_Iter	Random (Partition)	Climbing (Partition)	Annealing (Partition)
Mean	43.62	83.08	67.94
Min	0	0	0
Q1	12	27	26
Median	33	51	51
Q3	59	128	85
Max	183	460	428

There is so much data here that it would be impossible to say everything of note, but here are the major things we found:
1. Pre-partitioning is by far the best system, then a simple Karmakar-Karp, and then finally the sign assingment method is the worst. What's interesting about this is that the sign method is so bad, that one iteration of Karmarkar-Karp is ordered of magnitude better.
2. In all cases, the median residue is, give or take, about half the Q3 residue, and double the Q1 residue.
3. There is very little change, especially for the climbing signs from 25k to 100k. In fact (not shown in these tables), not a single of the 100 runs saw a different best value for climbing after 100k iterations than after 50k iterations.

4.1.2 Annealing and T(iter)

When reading the pset, I was a little surprised by something. As iter got bigger, $T(iter)$ got smaller, which in turn decreased the chances that any point the annealing would choose a new one. This doesn't make sense because as time goes on, you're more and more likely to be stuck. So in reality, $T(iter)$ should get bigger as time goes on. So, I change $T(iter)$ from being $10^10 * 0.8^{iter/300}$ to $10^101.3iter/300$.
Ultimately, this saw a large improvement in the signs annealing run. With the pset's $T(iter)$ function, as iter got very large annealing essentially just became summation and as a result plateaued off in exactly the same way. But it no longer does that.

5 Karmarkar-Karp as a Starting Point

For the sign algorithms, using Karmarkar-Karp as a starting point would be a large improvement. For the number of trials tested, none of the sign algorithms came close to the residues left by Karmarkar-Karp. Since all of the sign algorithms use the best partition they find, starting with Karmarkar-Karp will nearly guarantee an improvement.

Since the random partition algorithm gives better results than Karmarkar-Karp, and since the starting point has no effect on the other partitions tested, it is unlikely that starting with Karmarkar-Karp will improve the algorithm, and it is also unlikely to hurt the performance. For the hill climbing and simulated annealing partition algorithms, starting with Karmarkar-Karp is likely to hurt. This is because Karmarkar-Karp gives a partition of two subsets, and so there would be fewer similar partitions for the algorithms to explore. Thus, the algorithm will not be able to explore as much, hurting its ultimate performance.