CS 124 Programming Assignment 3: Spring 2022

Your name(s) (up to two): Burak Ufuktepe

Collaborators: None
No. of late days used on previous psets: 19
No. of late days used after including this pset: 23

Table of Contents

Abbreviations

DP : Dynamic Programming
HC : Hill Climbing
KK : Karmarkar Karp
NP : Non-deterministic Polynomial
PP : Prepartition
RR : Repeated Random
SA : Simulated Annealing
STD : Standard

Introduction

Consider the Number Partition problem where we have a sequence $A = (a_1, a_2, \dots, a_n)$ of non-negative integers such that the sequence of terms in A sum up to some number b. We would like to split A, into two subsets A_1 and A_2 such that the *residue* u is minimized:

$$u = \left| \sum_{y \epsilon A_2} y - \sum_{x \epsilon A_1} x \right|$$

This problem is NP-complete. However, there is a pseudo-polynomial time Dynamic Programming algorithm that solves this problem.

DP Solution to the Number Partition Problem

The *residue* of the Number Partition problem is minimized when we have $\sum_{x \epsilon A_1} x = \lfloor b/2 \rfloor$. So, u can be expressed as follows:

$$u = \left| \sum_{y \epsilon A_2} y - \sum_{x \epsilon A_1} x \right| = \left(b - \left\lfloor \frac{b}{2} \right\rfloor \right) - \left\lfloor \frac{b}{2} \right\rfloor = b - 2 \left\lfloor \frac{b}{2} \right\rfloor$$

If b is even u can be as low as 0. That is, $\left\lfloor \frac{b}{2} \right\rfloor = \frac{b}{2}$ so we will have:

$$u = b - 2 \left\lfloor \frac{b}{2} \right\rfloor = b - b = 0$$

If b is odd u can be as low as 1. That is, $\left\lfloor \frac{b}{2} \right\rfloor = \frac{b-1}{2}$ so we will have:

$$u = b - 2 \left\lfloor \frac{b}{2} \right\rfloor = b - 2 \frac{b-1}{2} = 1$$

Therefore, the problem becomes identifying a subset of A whose sum of elements is $\lfloor b/2 \rfloor$ or as close to $\lfloor b/2 \rfloor$ as possible.

Let $D[i, j]$ be a 2D boolean array that indicates whether there exists a subset of $T = (a_1, \dots, a_i)$ such that the sequence of terms in T sum up to j where $0 \leq i \leq n$ and $0 \leq j \leq \lfloor b/2 \rfloor$.

We initialize every element of $D[i, j]$ to $False$ and then we set $D[i, 0] = True$ for all $0 \leq i \leq n$. This is because all sets have a subset (namely the empty set) whose elements sum to 0.
Then we can populate the $D[i, j]$ array by iterating over each row i from 1 to n and in each iteration we iterate over each column j from 1 to $\lfloor b/2 \rfloor$. Note that we initialize every element of $D[i, j]$ to $False$ and we iterate over each i from 1 to n so we will always have $D[0, j] = False$ where $0 < j \leq \lfloor b/2 \rfloor$. This is because an empty set does not have a subset whose elements sum up to j.

To determine $D[i,j]$ we use the following recurrence relation:

$$D[i,j] = \begin{cases} D[i-1,j], & if\ a_i > j \\ D[i-1,j]\ or\ D[i-1,j-a_i], & otherwise \end{cases}$$

To calculate the *residue*, we first identify the largest j such that $D[n,j]$ is $True$. This value is essentially the sum of the elements of A_1, that is $\sum_{x \epsilon A_1} x = j$. Hence, the sum of the elements of A_2 will be $\sum_{y \epsilon A_2} y = b - j$. So, the *residue* can be calculated as follows:

$$u = \left| \sum_{y \epsilon A_2} y - \sum_{x \epsilon A_1} x \right| = (b-j) - j = b - 2j$$

Correctness: we show that our recurrence relation correctly calculates $D[i,j]$. That is, for a given value of i and j where $0 \leq i \leq n$ and $0 \leq j \leq \lfloor b/2 \rfloor$, it correctly identifies whether there exists a subset of $T = (a_1, \dots, a_i)$ such that the sequence of terms in T sum up to j.

The proof is by induction on i.

Base Case: when $i = 1$ and when we have an arbitrary value of j such that $1 \leq j \leq \lfloor b/2 \rfloor$, there are three cases:

> **Case 1:** if $a_1 > j$ that means there does not exist a subset of $\{a_1\}$ whose terms sum up to j. In this case, our recurrence relation sets $D[1,j]$ to $D[0,j]$. Since $D[0,j]$ is initialized to $False$ we will have $D[1,j] = False$, which is correct.
>
> **Case 2:** if $a_1 = j$ that means there exists a subset of $\{a_1\}$ whose terms sum up to j, namely the set $\{a_1\}$. In this case, our recurrence relation sets $D[1,j]$ to $D[0,j]$ or $D[0,j-a_1]$ where $D[0,j-a_1] = D[0,0]$. Since $D[0,0]$ is initialized to $True$ we will have $D[1,j] = True$, which is correct.
>
> **Case 3:** if $a_1 < j$ that means there does not exist a subset of $\{a_1\}$ whose terms sum up to j. In this case, our recurrence relation sets $D[1,j]$ to $D[0,j]$ or $D[0,j-a_1]$. Since both $D[0,j]$ and $D[0,j-a_1]$ are initialized to $False$ we will have $D[1,j] = False$, which is correct.

So, the base case holds.

Inductive Hypothesis: Assume that our recurrence relation correctly calculates $D[k,j]$ where $k < n$ and $1 \leq j \leq \lfloor b/2 \rfloor$.

Inductive Step: We show that our recurrence relation correctly calculates $D[k+1,j]$. In the $(k+1)^{th}$ iteration there are two cases:

> **Case 1:** if $a_{k+1} > j$ then we cannot include a_{k+1} in the solution as its value exceeds j. That is, we need to consider whether there exists a subset of $(a_1, \dots, a_k)$ whose terms sum up to j. By

the Inductive Hypothesis this was correctly calculated and stored in $D[k, j]$. Hence, we set $D[k+1, j]$ to $D[k, j]$.

Case 2: if $a_{k+1} \leq j$ then we consider the following two conditions:

- If there exists a subset of $(a_1, \dots, a_k)$ whose terms sum up to j, then we can use that subset as our solution. Hence, we can set $D[k+1, j]$ to $D[k, j]$. By the Inductive Hypothesis, $D[k, j]$ must be $True$. Therefore, $D[k+1, j]$ will be $True$.
- If there exists a subset of $(a_1, \dots, a_k)$ whose terms sum up to j-a_{k+1}, then we can use that subset and include a_{k+1} so that the sum of values will be $j - a_{k+1} + a_{k+1} = j$. Hence, we can set $D[k+1, j]$ to $D[k, j - a_{k+1}]$. By the Inductive Hypothesis, $D[k, j - a_{k+1}]$ must be $True$. Therefore, $D[k+1, j]$ will be $True$.
- If $D[k, j]$ and $D[k, j - a_{k+1}]$ are both $False$ then there exists no subset of $(a_1, \dots, a_{k+1})$ whose terms sum up to j. Hence, $D[k+1, j]$ will be set to $False$.

Since these are the only possibilities and, in each case, $D[k+1, j]$ is correctly calculated we conclude that our recurrence relation is correct.

Runtime: Our algorithm iterates over each row i from 1 to n and each column j from 1 to $\lfloor b/2 \rfloor$. And in each iteration, it does constant amount of work. Hence it runs in $O(nb)$ time.

Karmarkar-Karp in O($n$log$n$) Time

The KK algorithm can be implemented in $O(n \log n)$ time using a max-heap data structure. Let A be our input array that has n elements. We construct a max-heap using the elements in A. Then we do the following $n - 1$ times:

- Extract the largest two elements from the max-heap.
- Insert their absolute difference back into the max-heap.

Finally return the only element left in the max-heap.

Correctness: per the Karmarkar-Karp algorithm, in each iteartion we are removing the largest two elements remaining in the heap and inserting their absolute difference back into the heap. Hence, in each iteration we are reducing the number of elements in the heap by one. Therefore, after $n - 1$ iterations there must be one element left in the heap which is the *residue* according to the Karmarkar-Karp algorithm. Hence, we conclude that our algorithm follows the same steps as the Karmarkar-Karp algorithm and correctly outputs the *residue*.

Runtime: constructing the max-heap takes $O(n)$ time. Extracting the largest element from the heap and inserting an element to the heap takes $O(\log n)$ time. Since we iterate $n - 1$ times, the total runtime becomes $O(n \log n)$.

Results

50 trials are performed using randomly generated numbers in the range $[1, 10^{12}]$. In each trial, a new initial random solution is generated for the standard and prepartition representations. The same initial random solution is used for each of the standard heuristics in order to better compare their results. Similarly, the same initial random solution is used for each of the prepartition heuristics. The average residues and runtimes over the 50 trials for each algorithm are given in Table 1 and Table 2, respectively. Note that KK is not included in Table 2 as it is called only once (runtime is less than 1 millisecond) whereas the other algorithms performed 25000 iterations.

Table 1: Average Residues

KK	RR (STD)	HC (STD)	SA (STD)	RR (PP)	HC (PP)	SA (PP)
222099	247428676	296107916	274610090	169	618	231

Table 2: Average Runtimes (ms)

RR (STD)	HC (STD)	SA (STD)	RR (PP)	HC (PP)	SA (PP)
323	183	190	6283	5526	5736

The scatter plot given in Figure 1 provides us more insight on how each algorithm performed in each individual trial.

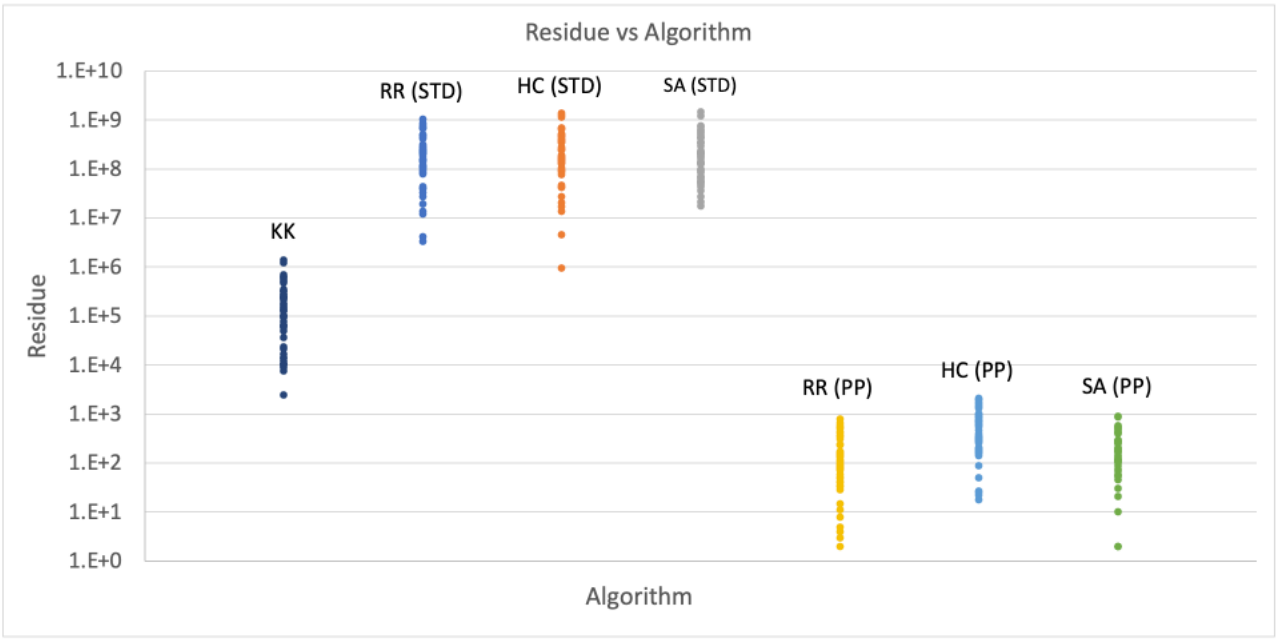

Figure 1: Results of 50 Trials

Runtimes

In terms of runtime, prepartitioned algorithms take longer than standard algorithms. This makes sense, because in each iteration, the prepartitioned algorithms derive a new sequence A' from A which takes $O(n)$ time (assuming arithmetic operations take constant time), and then run KK which takes $O(n \log n)$ time. Whereas, the standard algorithms multiply the elements of the solution array by the elements of the sequence A and sum up the results which takes $O(n)$ time (again assuming arithmetic operations take constant time).

When we look at the runtimes of RR, HC, and SA algorithms, for both representations we see that RR has the highest runtime and HC has the lowest runtime. This is because the Repeated Random algorithm generates a new solution in every iteration by calling the random number generator n times. Whereas, the Hill Climbing and Simulated Annealing algorithms call the random number generator at most a few times to generate a neighbor solution. Furthermore, the runtime of Simulated Annealing is slightly higher than the runtime of Hill Climbing. This is mainly due to the probability function that Simulated Annealing needs to compute whenever it comes across a worse solution.

Residues

The average residues in Table 1 and the scatter plot show that the prepartitioned algorithms yield much lower residues than standard algorithms. This is because the standard algorithms randomly partition the list of elements into two groups whereas the prepartitioned algorithms first partition the list of elements up to n groups and then run the KK algorithm. As a result, prepartitioned algorithms yield lower residues because KK produces better results compared to completely random solutions.

As we can see from the data in Table 1, RR performed the best and HC performed the worst among the three algorithms for both representations. This makes sense because RR has the highest flexibility among the three since it has the ability to test completely random solutions. On the other hand, HC has the issue of getting stuck in local minima. Based on our trials, SA sits in between RR and HC. Even though SA follows the same general approach as HC, it has the ability to escape from a local minimum with some probability hence it has better results compared to HC. Even though the average residue of RR is lower compared to SA, in some trials SA produced lower residues than RR. Since both are randomized algorithms, we cannot conclude one is better than the other. However, the trials show that, in general, both SA and RR perform better than HC.

In order to better understand how each algorithm behaves, for a sample trial, the lowest residue seen so far for each iteration is plotted for the standard and prepartitioned algorithms in Figure 2 and Figure 3, respectively.

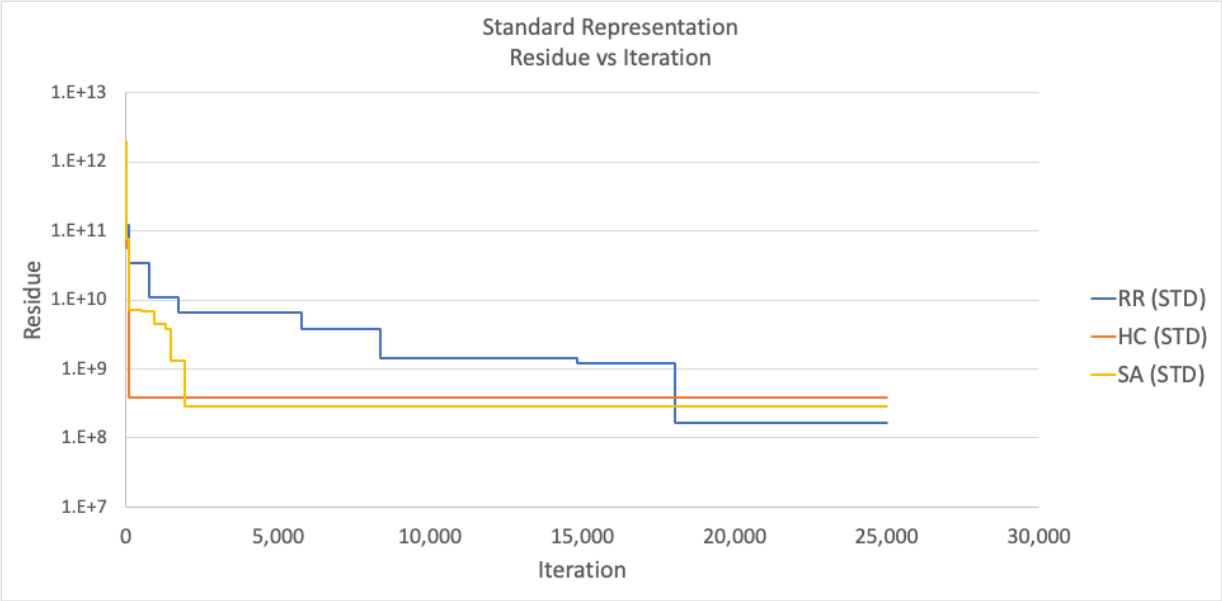

Figure 2: Standard Representation Residue Values for a Single Trial

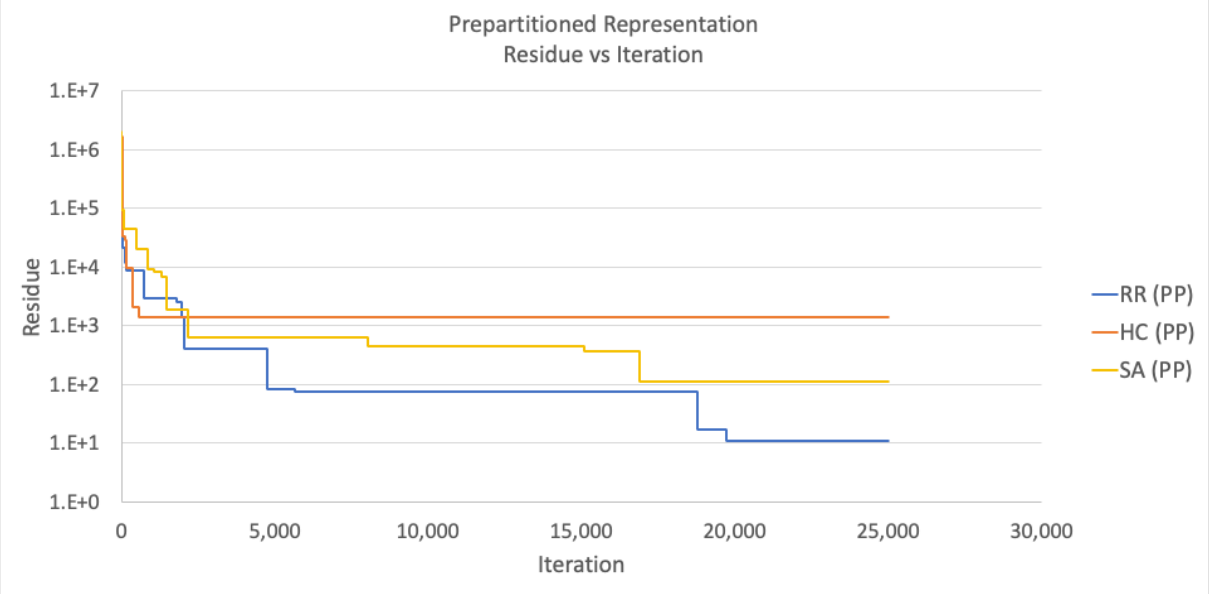

Figure 3: Prepartitioned Representation Residue Values for a Single Trial

In both graphs we see that after the initial iterations, the HC algorithms were not able to lower the residue. This shows they got stuck in a local minimum. We also observe that the SA algorithms performed better than the HC algorithms as expected. However, the SA algorithms also suffered from the issue of getting stuck in local minimum. This can be clearly seen in Figure 2 where the standard SA algorithm levels off after performing around 2500 iterations. Although, this effect is less obvious for the prepartitioned SA algorithm. This is because the prepartitioned SA algorithm produces lower residues. As a result, the probability of moving to a worse solution is higher compared to the standard SA algorithm. Hence, it has a higher chance of escaping from a local minimum. Finally, we observe that the RR algorithms were able to reduce the residue even at higher iterations since they have the flexibility of exploring completely random parts of the solution space.

Karmarkar-Karp as a Starting Point

Here we will consider the impact of using the solution produced by KK as a starting point on each algorithm separately:

Standard Repeated Random

In Figure 1 we see that the residues that standard RR produces are generally higher than KK residues. If we use the solution produced by KK as a starting point for standard RR, the final residue that standard RR produces will be at least as good as the KK residue. However, using the solution produced by KK will only impact the first iteration of standard RR because in each iteration the algorithm picks a completely random solution. Furthermore, the final residue that standard RR produces will most likely be the same as the residue produced by KK. Because, although possible, it is unlikely that the standard RR will be able to lower the residue produced by KK since the experimental results show that choosing a random solution in each iteration produces higher residues compared to residues produced by KK.

Standard Hill Climbing and Simulated Annealing

Again, if we look at Figure 1 we see that the residues that standard HC and SA produce are generally higher than KK residues. If we use the solution produced by KK as a starting point for standard HC and SA the final residues that standard HC and SA produce will be at least as good as the KK residue. However, although possible, it is unlikely that neither SA nor HC will be able to produce a residue lower than KK. We draw this conclusion from Figure 2, where both standard SA and HC got stuck at a local minimum even when the residue is greater than 10^8 whereas KK residues are generally at most $\sim 10^6$. Therefore, there is a good chance that the KK starting point would be the best solution that standard SA or HC will find.

Prepartitioned Repeated Random

In Figure 1 we see that the residues that prepartitioned RR produces are lower than KK residues. Therefore, we conclude that using the solution produced by KK as a starting point for prepartitioned RR will most likely not have an impact on the results. This is because in each iteration prepartitioned RR picks a completely random solution and based on the experimental results we see that eventually it will most likely find a better residue than the residue produced by KK.

Prepartitioned Hill Climbing

We will show that prepartitioned HC will most likely not be able to reduce the residue produced by KK.

If we use the solution produced by KK as a starting point for prepartitioned HC, we will be starting with 2 groups, A_1 and A_2. Let $X = \sum_{x \epsilon A_1} x$ and $Y = \sum_{y \epsilon A_2} y$. Without loss of generality assume $Y \geq X$. Furthermore, let u be the residue produced by KK such that $u = Y - X$.

In each iteration, prepartitioned HC will generate a neighbor solution by removing an element a_i from one of the groups and place it into the other group with probability $1/n$ or into a new group with probability $(n-1)/n$. Hence, we consider the high probability case where a_i is placed into a new group.

a_i can be removed either from A_1 or A_2.

Case 1: If a_i is removed from A_1 then prepartitioned HC will run KK on the sequence $(X - a_i,\ Y,\ a_i)$. Since we have $Y > X - a_i > a_i$, KK will return:

$$Y - (X - a_i) - a_i = Y - X + 2a_i = u + 2a_i > u$$

So, the residue will not decrease.

Case 2: If a_i is removed from A_2 then prepartitioned HC will run KK on the sequence $(X,\ Y - a_i,\ a_i)$. We assume $a_i > u$ because in Figure 1 we observe that KK residues are generally at most $\sim 10^6$ whereas a_i will be larger than 10^{11} with 90% probability since $1 \leq a_i \leq 10^{12}$. Hence, we assume:

$$a_i > u$$

$$a_i > Y - X$$

$$X > Y - a_i$$

So, we have $X > Y - a_i > a_i$, therefore KK will return:

$$a_i - \big(X - (Y - a_i)\big) = a_i - X + Y - a_i = u$$

So, the residue will not decrease.

Hence, we conclude that there is a good chance the KK starting point would be the best solution that prepartitioned HC will find.

Prepartitioned Simulated Annealing

If we use the solution produced by KK as a starting point for prepartitioned SA, unlike prepartitioned HC, prepartitioned SA will probably find a better residue than the residue produced by KK. This is because with some probability the algorithm will move to a worse solution and it will eventually find a better solution than the solution produced by KK. Therefore, we conclude that using the solution produced by KK as a starting point for prepartitioned SA will most likely not have a large impact on the results.