CS124: Programming Assignment 3 Report

Christy Jestin & Tarek Aloui

Late Days Used: 0

Christy: Total Late Days Used: 9

Tarek: Total Late Days Used: 12

---

1 DP Solution to the Number Partition Problem

1.1 Algorithm

Inputs: A

Output: min_resid (minimum residual)

```
• b = sum(A)
• Let $arr$ be a boolean array of size $b + 1$.
• Set $arr[0] = 1$
• Let $min\_resid = \infty$
• For i from 0 to len(A)-1 (inclusive):
    – For j from 0 to b (inclusive):
        ∗ If $arr[j] == 1$:
            · c = j+A[i]
            · $arr[c] = 1$
            · Let $resid =| b - 2c |$
            · Set $min\_resid = min(min\_resid, resid)$
• Return $min\_resid$
```

1.2 Correctness

In this algorithm, arr is an array whose indices denote the possible sums of subsets of A. The sum of any subset of A is an integer between 0 and b since it is given that all inputs are nonnegative integers. Additionally $arr[s] = 1$ if and only if we've seen some subset of A that sums to s. $arr[0]$ is always 1 since the empty set sums to 0.

Let A_1 and A_2 be the two subsets that we divide A into. For each index i, we consider the subsets of A made up of only the first i elements. We've already considered the subsets made up of only the first $i - 1, i - 2, \ldots, 1$, and 0 elements, so we only need to consider the subsets that actually include the ith element. We calculate the sums of these subsets by iterating through arr and adding $A[i]$ to each index j where $arr[j] = 1$. This makes sense since $arr[j] = 1$ indicates that there already exists some subset that sums to j and uses only the first $i - 1$ elements. Thus adding the ith element to one of these subsets will create a subset that sums to $j + A[i]$. We calculate the residual by utilizing the fact that the sum of A_1 and the sum of A_2 must add up to b. Thus if A_1 sums to s, then A_2 sums to $b - s$, and the residual is $| (b - s) - s |=| b - 2s |$. We consider the sums of all possible subsets of A, and we keep track of the minimum residual, so the result is indeed the minimum possible residual, and we have correctly solved the Number Partitioning problem.

1.3 Complexity

We prove that the algorithm runs in $O(nb)$ with n being the number of elements in A and b being the sum of elements in A.

We begin by summing all elements to find b in $O(n)$ time. We iterate through all the elements (so n iterations) and at each element we consider all of the possible sums up to that element (at most $b+1$ different sums). Thus the overall algorithm runs in time $O(nb)$.

2 How Karmarkar-Karp's algorithm can be implemented in $O(n \log n)$ steps

First, we'll note a few invariants about the algorithm. At every step, we replace a pair of nonnegative integers with the absolute value of their difference and zero. The array of elements is non negative, so even if we put 0 back into the array, it won't ever be selected as one of the top two elements until the number of nonzero elements is less than or equal to 1, and this is the point where the Karmarkar-Karp algorithm will terminate. Thus we don't actually need to put the 0's back into the array, and it suffices to just insert the absolute value of the difference. Thus at every step in the algorithm, we replace a pair of integers with the absolute value of their difference, and the number of elements in the array decreases by 1 at each step. Thus we need to run $O(n)$ steps to partition the whole array.

Now we'll discuss the algorithm. We begin by inserting all elements into a max heap. This takes time $O(n \log n)$ as shown in class. Using a max heap allows us to grab the two largest elements in time $O(\log n)$ since it takes time to max-heapify after extracting the max. It also takes time $O(\log n)$ to insert the absolute value of the difference back into the heap. Thus each step in the Karmarkar-Karp algorithm takes time $O(\log n)$, and both initializing the max heap and running the algorithm can be done in time $O(n \log n)$. Note that this analysis assumes that it takes constant time to compute the absolute value of the difference.

3 Comparison of the different Number Partitioning Algorithms

3.1 Data

Table 1: Summary Statistics of the residuals obtained by the different Number partitioning Algorithms

Algorithm	mean	std	min	25%	50%	75%	max
Repeated Random	2.25E+08	1.98E+08	8.23E+06	8.55E+07	1.66E+08	2.92E+08	8.25E+08
Hill Climbing	2.96E+08	2.75E+08	1.93E+06	9.57E+07	2.50E+08	3.81E+08	1.21E+09
Simulated Annealing	2.20E+08	2.49E+08	8.68E+06	5.53E+07	1.34E+08	2.73E+08	1.31E+09
Karmarkar-Karp	4.51E+05	5.32E+05	2.61E+02	3.15E+04	1.52E+05	8.27E+05	1.97E+06
Prep. Repeated Random	8,782.22	25,566.17	0.00	27.50	71.00	6,575.25	159,408.00
Prep. Hill Climbing	18,582.54	49,743.72	3.00	66.50	227.50	4,359.50	220,311.00
Prep. Simulated Annealing	8,043.06	17,316.87	1.00	42.25	116.00	1,495.25	64,234.00

3.2 Graphs

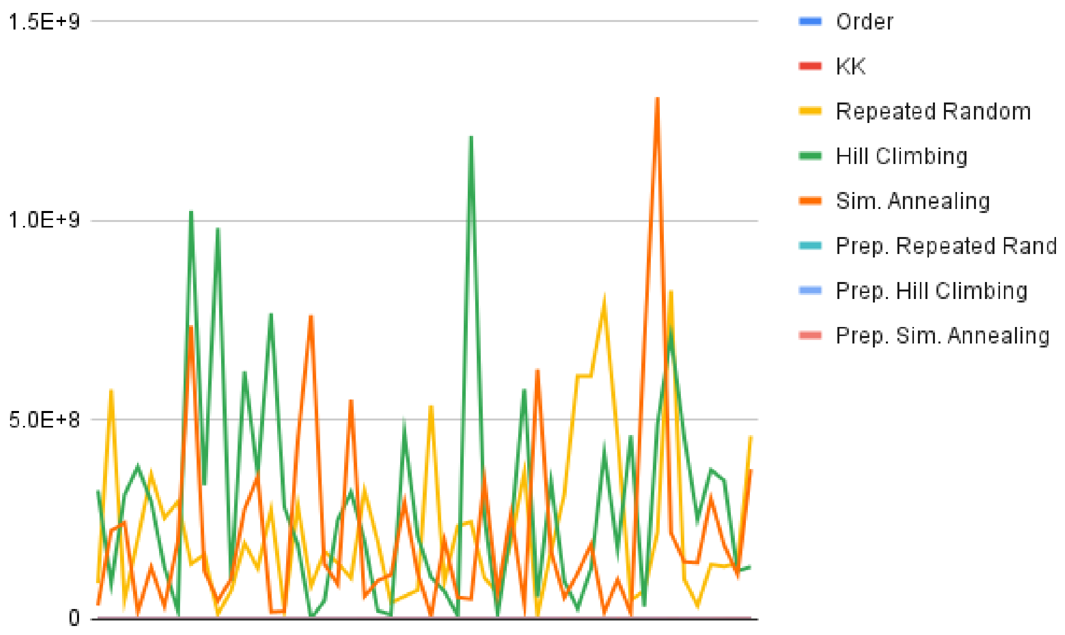

Figure 1: Line Plots of all Number Partitioning Algorithms combined

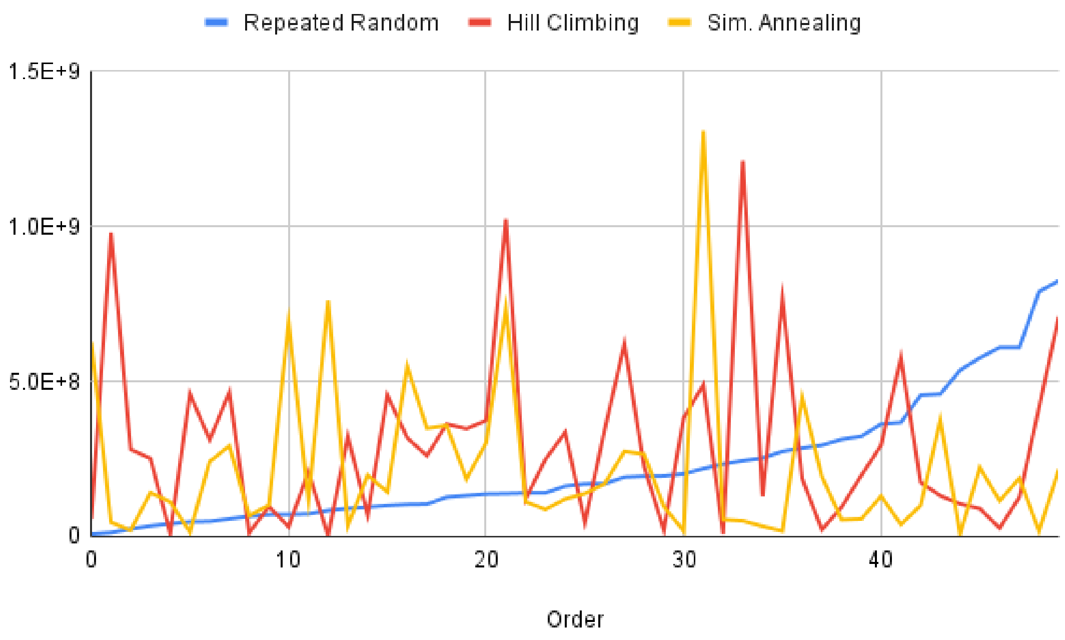

Figure 2: Line Plots of Regular Number Partitioning Algorithms ordered by Repeated Random

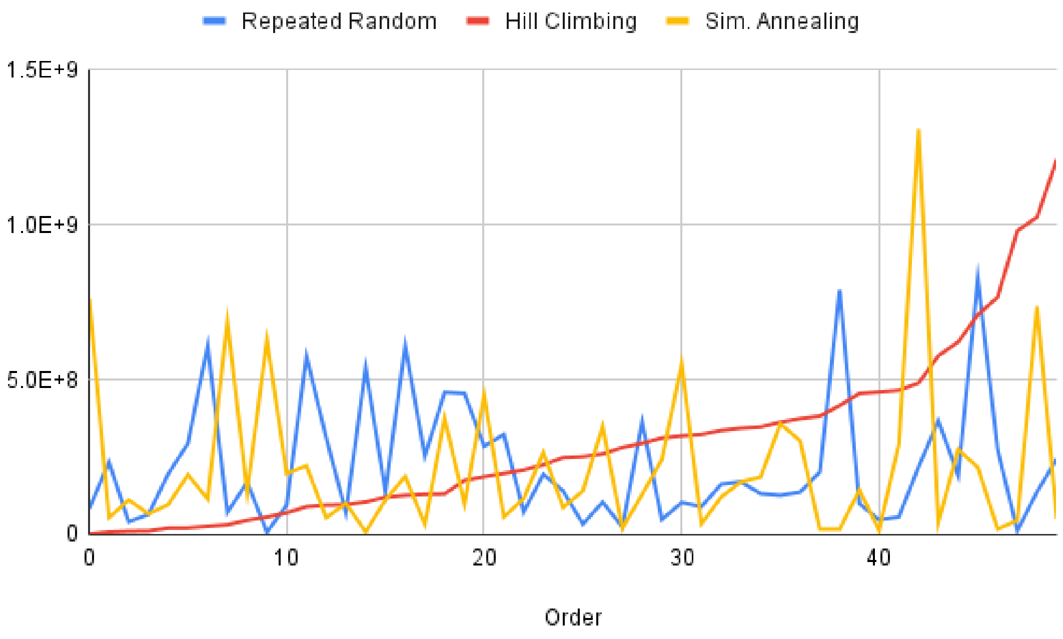

Figure 3: Line Plots of Regular Number Partitioning Algorithms ordered by Hill Climbing

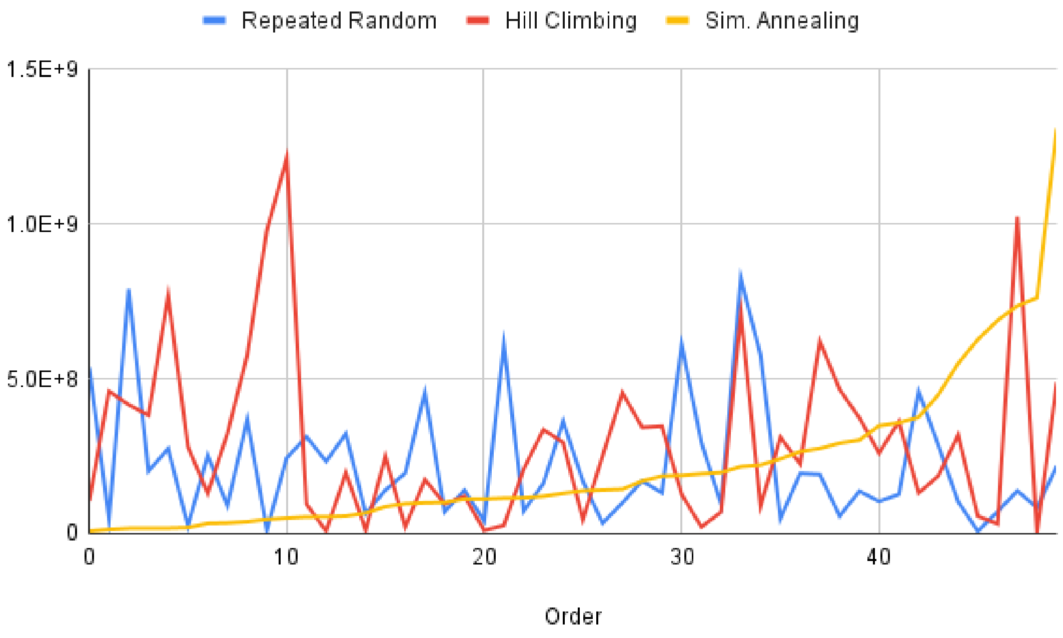

Figure 4: Line Plots of Regular Number Partitioning Algorithms ordered by Simulated Annealing

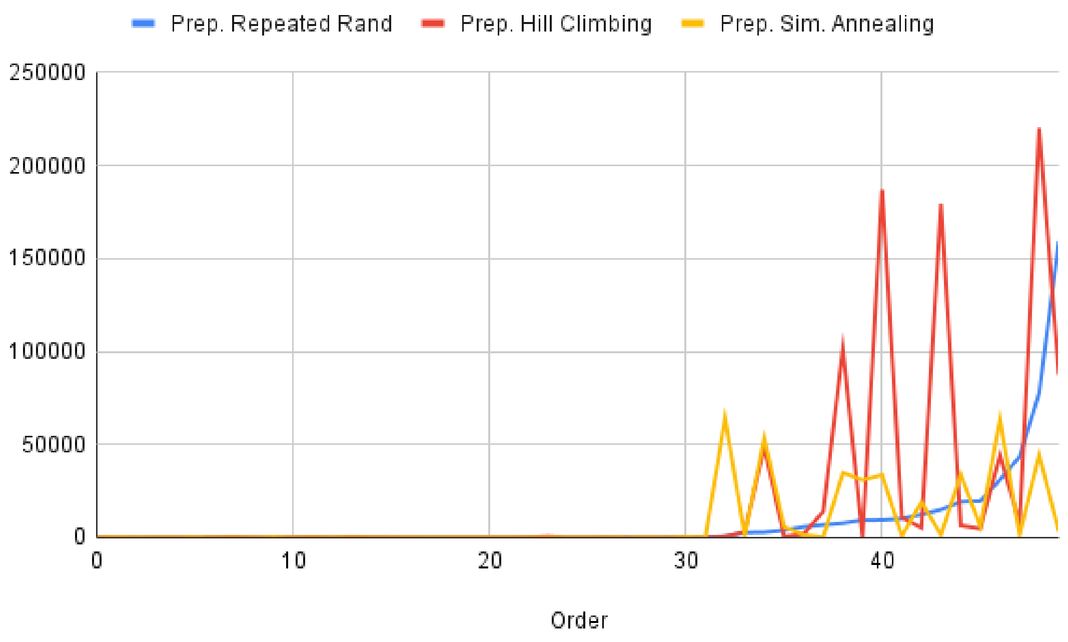

Figure 5: Line Plots of Pre-partitioned Number Partitioning Algorithms ordered by Pre-partitioned Repeated Random

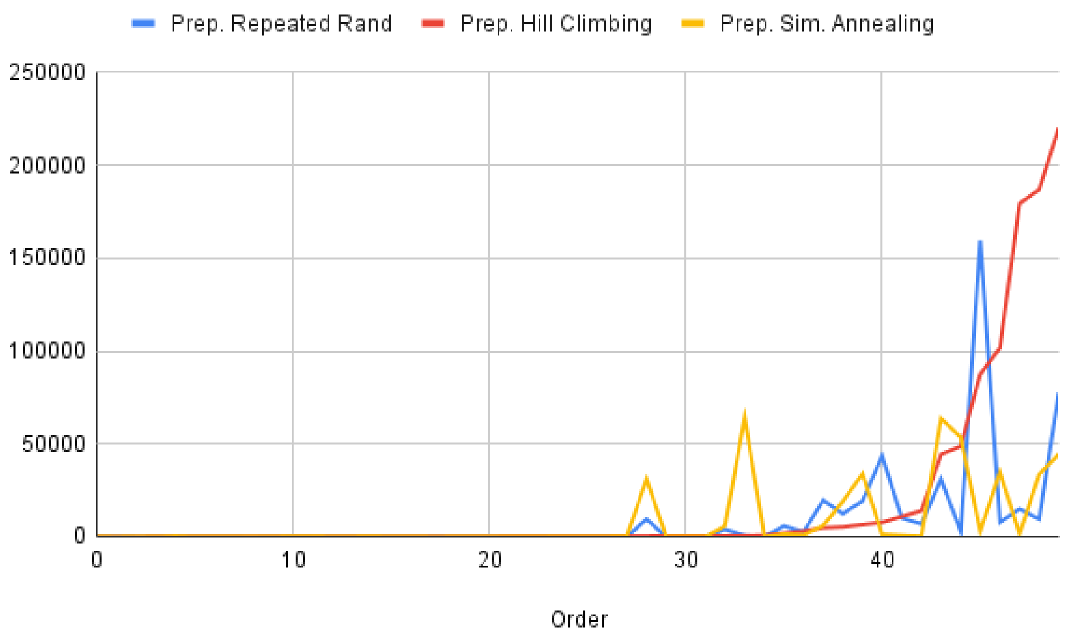

Figure 6: Line Plots of Pre-partitioned Number Partitioning Algorithms ordered by Pre-partitioned Hill Climbing

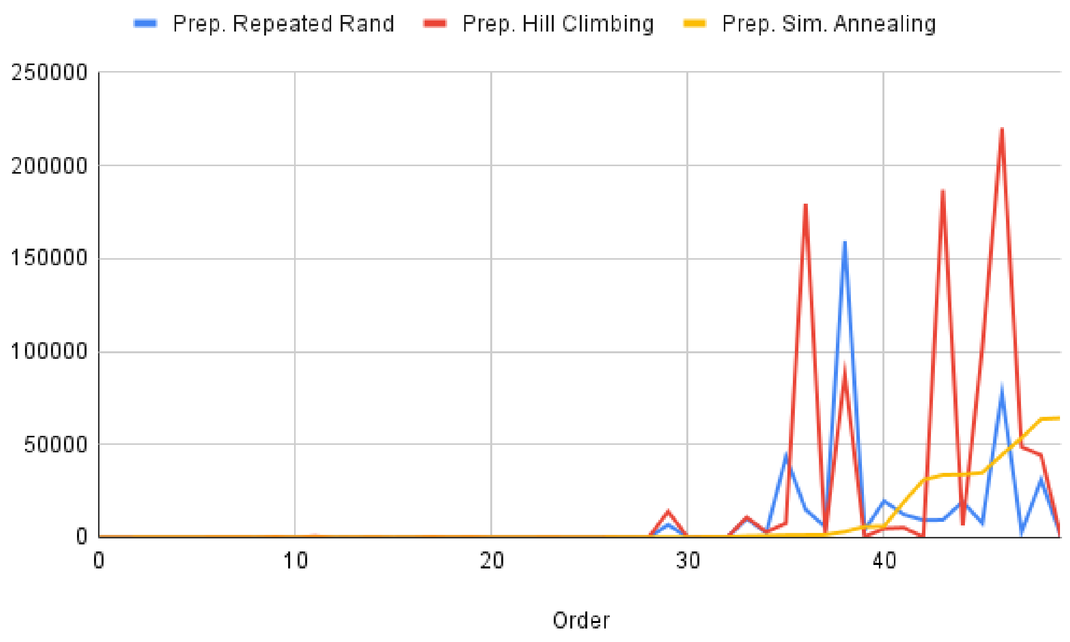

Figure 7: Line Plots of Pre-partitioned Number Partitioning Algorithms ordered by Pre-partitioned Simulated Annealing

3.3 Analysis

As we can see from the summary statistics above, the prepartitioned and regular representation algorithms have very different outputs, and the gap isn't even close. Across all inputs, the standard Karmarkar-Karp algorithm does better than any of the search algorithms with the regular representation but worse than any of the search algorithms with the prepartitioned representation. We can also see that the hill climbing algorithm generally does worse than the repeated random or simulated annealing algorithms when we use the same representation for all 3. Despite this, the solutions of all three algorithms are around the same order of magnitude when we fix the representation.
We can interpret these results by considering a few factors.

1. **Prepartitioning vs Regular Representation**

   To us the very clear gap between the two representations wasn't easy to explain. In the case of the repeated random algorithm, we can conclude that taking a random prepartition and then running Karmarkar-Karp is a better method than randomly choosing a full assignment. Choosing a full assignment is like taking a shot in the dark, and the prepartitioning method is like taking a few random steps and then doing something that has some intuition and reasoning behind it i.e. using Karmarkar-Karp.

   However, the same reasoning does not extend to Hill Climb or Simulated Annealing where we try to move better neighbors. One explanation is that our definition of neighbors is just more useful for finding better solutions in the prepartitioned representation than the regular representation since the definition of neighbors is somewhat arbitrary. For example, flipping just one assignment will often make the residue worse if we move an element from the smaller set to the bigger set. Thus many of 25,000 iterations could be wasted on exploring neighbors that aren't improvements. It could also be explained by the fact that moving one or two elements around can have very little effect on the residue when we're dealing with a list of 100 elements.

   While the neighbors in the prepartitioned representation only differ in one value, a new prepartitioning means a new input to the Karmarkar-Karp algorithm, and we don't have a great intuition for how big of an impact that the change in input has on the residue. It could create fundamental differences in the separation process that has lots of downstream effects within the algorithm or it could just be switching the input from $[100, 300, 1, 3]$ to $[100, 300, 2, 2]$ i.e. very little or no change.

2. **Hill Climbing vs Repeated Random and Simulated Annealing**

   The main flaw of the hill climbing algorithm is that it can get stuck in a local optimum. Simulated annealing solves this issue by allowing the algorithm to move to a worse neighbor with some small probability, and the problem just does not have this issue since we're choosing a random solution every time instead of moving to neighbors. It is surprising that repeated random is able to do about as well as the simulated annealing algorithm when the former just chooses random solutions and the latter tries to move towards better solutions, but this can be explained by two factors. First, our definition of neighbors may not be that useful as we noted earlier since if there's little connection between neighbors (for prepartitioned representation), then moving to neighbors might be about as good as choosing random solutions. Second, our search algorithms might be making careful steps that slowly move to good solutions, and the repeated random algorithms might be able to "cover more ground" by trying all sorts of different solutions.

3. **Relationships between Number Partitioning Algorithms for Same Input**

   In the graphs above, we tried to determine if there was some correlation between the different Number Partitioning algorithms and we tried to see if we can isolate some pattern that relates between them besides the fact that pre-partitioned algorithms generally have smaller residuals than KK, which also has residuals much smaller than the regular algorithms. Because of the different orders of magnitude for the residuals in the Pre-partitioned and Regular algorithms, we decided to compare the algorithms within these two categories separately rather than combining all of them. Notice that there is no clear pattern in figure 1 which has line plots of all the algorithms combined in one graph.

   In figures 2, 3 and 4, we order the table of residuals for the regular algorithms by the values obtained by Repeated Random, Hill Climbing and Simulated Annealing respectively. The reason we do this is that, if a correlation exists between two algorithms, then we would have both of them generally increasing in a similar manner. What we find, however, is that no such relationship exists. For example in figure 2, the repeated random algorithm has residuals in increasing order, while none of the other algorithms follow this increasing patter and oscillate unpredictably.

   A similar analysis was done for the pre-partitioned algorithms in figures 5, 6 and 7. We also find that there is no correlation between the 3 algorithms.

4 How to use Solutions from KK algorithm as a starting point

In each of our randomized algorithms, we choose a starting assignment at random before searching for better solutions. We can utilize the solution given by the Karmarkar-Karp algorithm as our starting point instead. Note that we can get the actual pair of sets from the Karmarkar-Karp algorithm by using the 2-coloring graph method described above. In the case of the standard representation, we use $+1$ for all elements in one set and -1 for all elements in the other set. In the prepartitioned representation, we'll use 1 for all elements in one set, and 2 for all elements in the other.
Starting at the Karmarkar-Karp solution might cause us to get stuck in a local optimum for the hill climb algorithms since we only move to a neighbor if it is an improvement. However, this is true regardless of how we choose the starting point. Since we only move to better neighbors, we can guarantee that the output of the hill climb algorithm is at least as good as the Karmarkar-Karp solution.
In the case of the repeated random algorithms, using the Karmarkar-Karp solution as the starting point doesn't affect our search since we're choosing random solutions at each iteration instead of moving to neighbors. Rather this would simply guarantee that the solution found by the repeated random algorithm is also at least as good as the Karmarkar-Karp solution. For the simulated annealing algorithm, we are less likely to get caught in a local optimum than in the hill climb algorithm since we sometimes move to a neighbor even if it's a worse solution. As before, the output of the simulated annealing algorithm is at least as good as the Karmarkar-Karp solution since we return the best solution we've seen so far.