Determining Critical Regions to Empirically Verify Thresholds within the Partitioning Problem

Scott Kerlin
Computer Science Department
University of North Dakota
Grand Forks, ND 58202
scott.kerlin@und.nodak.edu

Dr. Thomas O'Neil
Computer Science Department
University of North Dakota
Grand Forks, ND 58202
oneil@cs.und.edu

Abstract

For NP-Complete problems, the threshold region represents where the difficult instances exist. For the partitioning problem it has been proposed that that the key threshold is located at *m/n* where *m* denotes the average number of bits needed to represent each integer in the list, and *n* denotes the number of integers in the list. The examination of this thresholding behavior, along with run-time consequences, is the focus of this research.

To generate data for this research, several instances of the partitioning problem were run against the "fast" algorithm proposed by Richard Stearns and a naive back-tracking algorithm to produce step count data. By examining these two divergent algorithms as their data passes through the critical thresholding region, we are better able to illustrate what is happening within the critical region and why.

Additionally, we discuss how space limitations affect the generation of empirical data.

1 Introduction

A number of claims regarding threshold limits have been made by theorists with respect to NP-Complete problems. In particular, there is the claim put forward by Ian Gent and Toby Walsh that the key threshold for the partitioning problem is located at m/n where m denotes the average number of bits needed to represent each integer in the list, and n denotes the number of integers in the list [4]. While purely mathematical and physics based proofs have been put forward supporting this claim (most notably by Christian Borgs, Jennifer Chayes, and Boris Pittel [5,6]), empirical data demonstrating this relationship is difficult to find. The goal of this research then, is to generate, collect and analyze empirical data to support the claim of the where the thresholding phenomena for the partitioning problem exists and to use the resulting information to illustrate how that threshold affects run-time.

2 Background

The observation of thresholding behavior over a given problem space provides insight into the problem being study. For a NP-Complete problem space, the threshold region represents where the difficult instances of the specific problem can be found [1]. Specifically speaking, the threshold is located where we move from the overconstrained region to the underconstrained region of the given problem space [1]. This concept is fairly easy to illustrate with the traditional NP-Complete graph. You have the cases where the graph is dense (overconstrained) or where the graph is sparse (underconstrained). It should be obvious that to move between this two states is a simple matter of adding or removing edges to the graph.

2.1 Thresholding and Partition

The problem with how to apply thresholding to the partitioning problem starts with with identifying what parameter can be manipulated to move from overconstrained to underconstrained (or the reverse). For the purposes of this discussion, consider the partition problem to be quite generalized. The goal of the partitioning problem is to partition a list of integers into two equally valued partitions.

To illustrate how how the partitioning problem would then look at its two states, consider the a few extreme instances of the partitioning problem. Suppose we have an list containing 1000 integers. Suppose also that all of these integers are the value 1. This should be a trivial problem to solve since to arrive at a perfect partition all that is needed is to count [2]. There exist a very large number of acceptable partitions for this particular type of list, and it is obvious to be a very trivial problem to arrive at any one of those possible solutions.

Consider next the creation of a list containing only two integers and attempting to partition that list. Clearly this problem is simply the likelihood of any two chosen integers being equal to each other [2]. These extreme examples illustrate quite well property of the two regions where on one extreme we have a large number of solutions and on the other extreme we have very few solutions [1].

2.2 Phase Transition

The exact moment within a problem space at which an instance of the problem moves from one of the two regions into the other, is called the phase transition. The phase transition carries with it the property that the phase transition occurs at the point where the solution probability changes from almost zero to almost one [1]. This is important to remember since it shows that once we pass the phase transition, we begun to run into the harder instances of our problems.

Since the phase transition is so important, we need to know exactly where within the problem space the phase transition falls. To locate this phase transition with in the partitioning problem we first need to determine what the parameter is which controls it. Since partition deals with a list of numbers, we have two obvious places to look for this parameter. One place to look for the phase transition is as the number of elements in the list is increased. The other obvious choice would be the number of bits required to represent the list in memory.

It turns out that both are right ideas are right. If m represents the average number of bits needed to store the value of each integer in the list, and n represents the number of integer in the list, then the ratio m/n determines where the critical region within partitioning problem's problem space [2]. Recalling the examples from section 2.1, we notice that when m/n is very small, then we many solutions. When m/n is very big, then we have very few solutions. The work of Christian Borgs, Jennifer Chayes, and Boris Pittel placed the value for the phase transition to be at $1 = m/n$ [5,6]. So where $m/n < 1$, there are likely to be many solutions to the instance, and where $m/n > 1$, there are likely to be few, if any, solutions.

3 Generating Empirical Data

Knowing where the phase transition is, the generation of data to illustrate the phenomena as we approach the threshold will need to broken into several parts. Since the number of elements is one factor, data should be generated for the case where the average bit size of the integers is held constant, and the number if integers in the list is varied. As the other factor is the average size of each integer in the list, data should be generated for the case where the number of integers in the list is held constant and the average bit size of the integers is varied. Then date will need to be generated for the case where both integer

size and number of integers are varied while holding the ratio between the two contributors is held constant.

With the testing cases established, the next step in the process is selecting what kind of partitioning algorithm(s) to use. An obvious choice to select is a naive backtracking algorithm to see if thresholds appear where they predicted to be for simple algorithms. Selecting a "higher" grade algorithm to compare the naive backtracking algorithm with is trickier. Polynomial heuristics could be included, but they don't guarantee that a solution will be found if one exists. This makes these algorithms less interesting for this study. If one was curious about thresholding and critical regions using heuristics, there is a good empirical study using the Karmarkar-Karp algorithm [7]. So with heuristics removed from the selection process, the "fast" Stearns algorithm was selected as a more advanced partitioning problem algorithm to compare with the naive backtracking algorithm as it is generally accepted as an efficient algorithm [8].

To create lists of appropriate properties for use turned out to be an interesting problem in itself. Since keeping track of the average bit size of the integers within the list is vital for this research, many lists generated (using Java's random method), ensuring that each average bit size / element list size combination eventually managed ten thousand trials actually took almost as much processor time as running the partitioning algorithms. This is because selecting n elements of very small bit sizes is very hard when the selection range across an even distribution with a high max value. This problem was introduced as a result of keeping the range of integers available to be selected into the list constant over the data collect. Doing so allows for a better picture of the problem than simply changing the max value allowed for a given average bit size since we can now observe lists like 1, 5, 7, 1, 1, 3, 2, 6 to have an average bit size of 2, where as having a hard max value cap of 3 (largest 2 bit number) would have exclude this type of list.

4 Analysis

With data generation and collection complete the investigation into can be learned from it.

4.1 Transition Phase Verification

The phase transition has an expected outcome based on prior works already referenced, so it seems a logical place to begin analysis. Starting with the data for where the number of integers was held constant at 15, and the average bit size of the integers in the list was varied Figure 1 was produced from the "fast" Stearns algorithm and Figure 2 was produced from the backtracking algorithm. These figures display the percentage of successfully solved partition instances as the average bit size of the integers changes. As n is held constant at 15 elements, where $m < 15$, there are likely to be many solutions to

the instance, and where m >15, there are likely to be few, if any, solutions. Judging by the figures, it appears that this relationship has been verified.

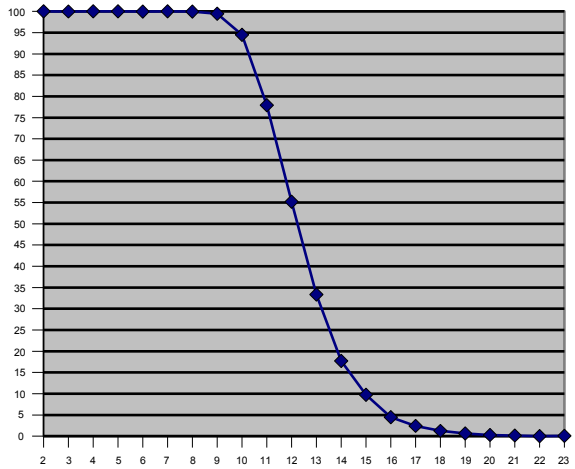

Figure 1: Percentage of Successfully Solved Instances as Average Bit Size Varies (Stearns)

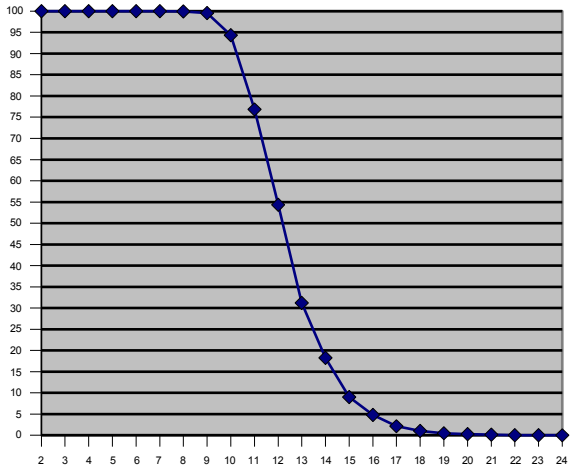

Figure 2: Percentage of Successfully Solved Instances as Average Bit Size Varies (Backtracking)

Examining the data produced where m is held constant and n is permitted to vary, Figure 3 and Figure 4 are produced (for the "fast" Stearns and backtracking algorithms respectively) to display the percentage of successfully solved partition instances. In this set of figures m is held constant at an average integer bit size of 15. This means that in instances where n > 15, there are likely to be many successful solutions to the instance, and where n <15, there are likely to be few, if any, successful solutions. Once again the data supports the theoretical claims. It should also be noted that both algorithms are reporting similar percentages of successful solutions, which shows that they both are exhibiting the same level of accuracy.

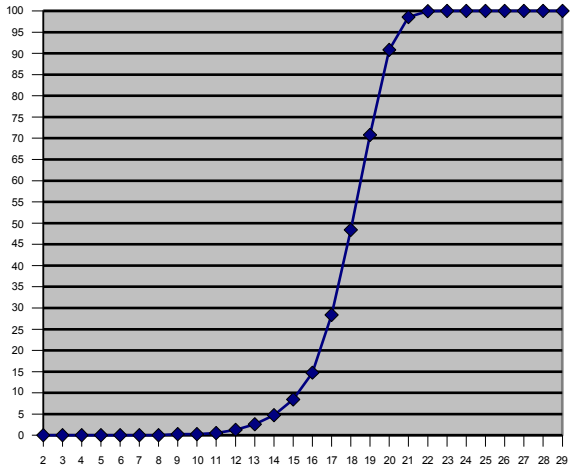

Figure 3: Percentage of Successfully Solved Instances as Number of Integers Varies (Stearns)

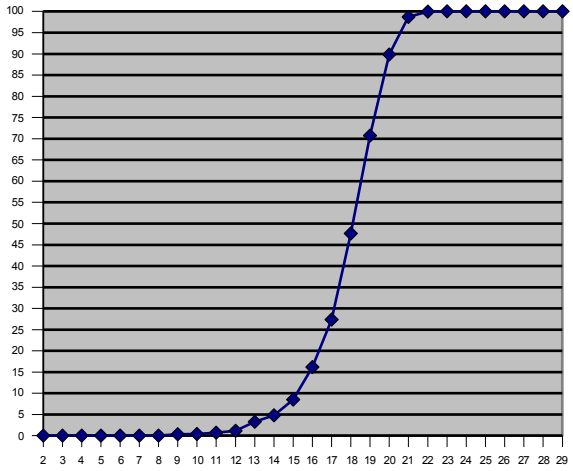

Figure 4: Percentage of Successfully Solved Instances as Number of Integers Varies (Backtracking)

4.2 Thresholding/Run-Time Analysis

Having established that the underlying theory is sound, attention may be turned to how this theory relates to running-time. Since straight out CPU would be affected by the hardware platform, step counts were used instead. It should also be noted, which will be addressed later in the research, that the "fast" Stearns algorithm suffers from a significant space overhead which prevents it from handling as large of a problem instance size on the hardware/language combination used to generate the data.

Figure 5 (Stearns) and Figure 6 (backtracking) are generated from step count data as *m* varies while *n* is held constant at 15 elements. These figures reveal some interesting behaviors. The backtracking algorithm displays a clear thresholding behavior as *m* increases past n, while the Stearns algorithm starts to exhibit the same behavior, but with an step count spike associated with it. As we consider the way in which the two algorithms work, we can begin to see why this is the case.

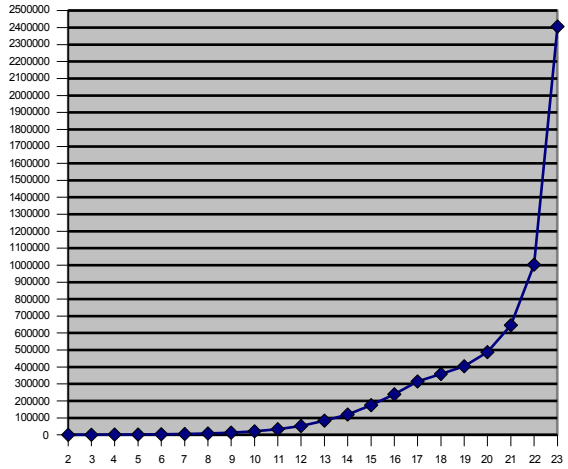

Figure 5: Step Count as Average Bit Size Varies (Stearns)

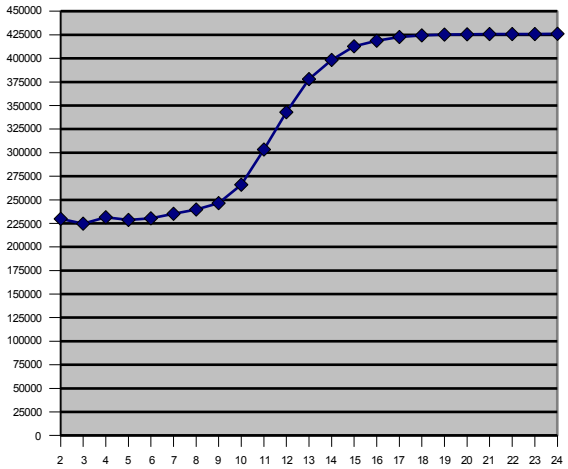

Figure 6: Step Count as Average Bit Size Varies (Backtracking)

First looking at the similarities, they both exit quickly in the region where there are a lot of possible solutions and exit slowly slowly in the region where there a few (if any) solutions. The backtracking algorithm tends toward a thresholding phenomena because its run-time is based on the number of recursive calls needed to check every possible combination for a given instance of the problem. The Stearns algorithm does have this thresholding phenomena because it requires an array equal to sum of the entire list minus the minimum element [8]. So as *m* increases, it causes the the algorithm's step count to

increase as well (to create and initialize the array). What makes the Stearns algorithm interesting is how the logarithmic graphs of the step counts show a distinct change in the growth pattern around the critical region (displayed in Figures 7 and 8).

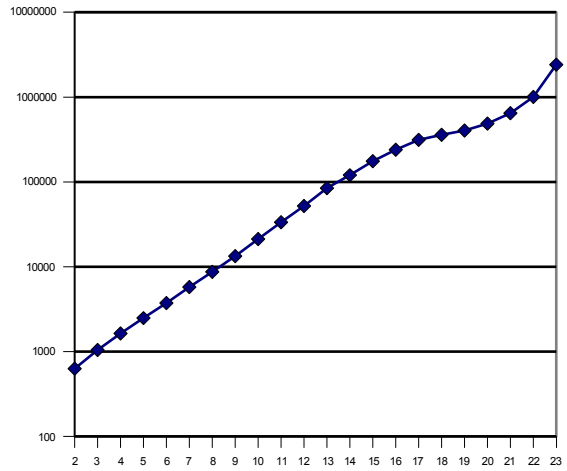

Figure 7: Logarithmic Step Count as Average Bit Size Varies (Stearns)

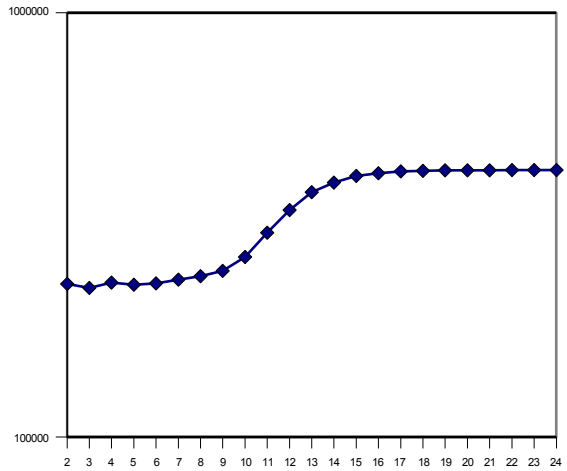

Figure 8: Logarithmic Step Count as Average Bit Size Varies (Backtracking)

Taking a look at the step counts graphs for the case where *m* is held constant and *n* varies, we encounter some interesting behavior as well (Figures 9 and 10 for Stearns and backtracking respectively). The step count graphs for both algorithms appear to be exponential at first glance, but because of the enormity of the scale differences, there could be some masking going on. So taking a look at the logarithmic versions of those graphs (Figures 11 and 12 for Stearns and backtracking respectively) a better picture of what is happening is revealed. Backtracking displays only a small change around the critical region threshold when examined from this perspective. This behavior is indicative of an inefficient algorithm since very little improvement is seen as the naive backtracking algorithm enters the solution plentiful region. Since the backtracking

algorithm is naively testing each combination, the algorithm is relying mostly on blind luck and random chance to terminate early [9]. The Stearns algorithm displays a far more interesting behavior. This algorithm insures high overhead for small element lists, so the data over the first six points is not surprising. What is interesting is that the logarithmic graph for the Stearns algorithm seems to display the same shape and behavior as the backtracking algorithm did in Figure 8 (after accounting for the initial overhead costs). Clearly the Stearns algorithm is locating solutions much more quickly in the solution plentiful region of the phase transition threshold. This sharp change within the critical region shows that the Stearns algorithm is quite efficient.

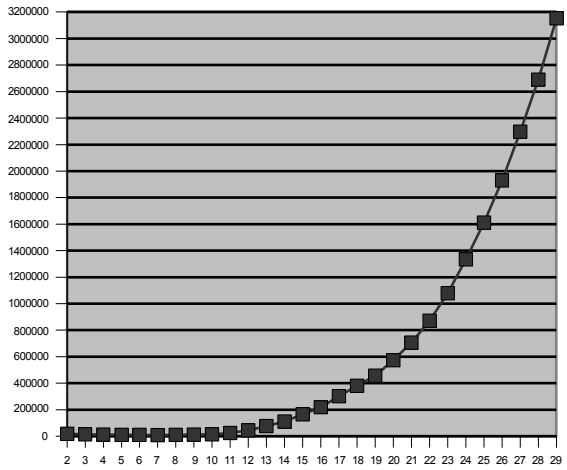

Figure 9: Step Count as Number of Elements Varies (Stearns)

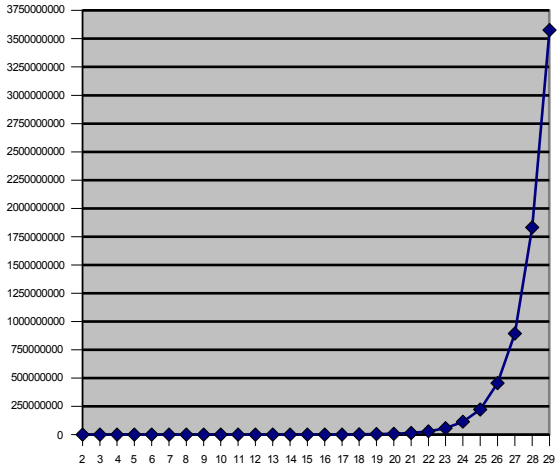

Figure 10: Step Count as Number of Elements Varies (Backtracking)

Figure 11: Logarithmic Step Count as Number of Elements Varies (Stearns)

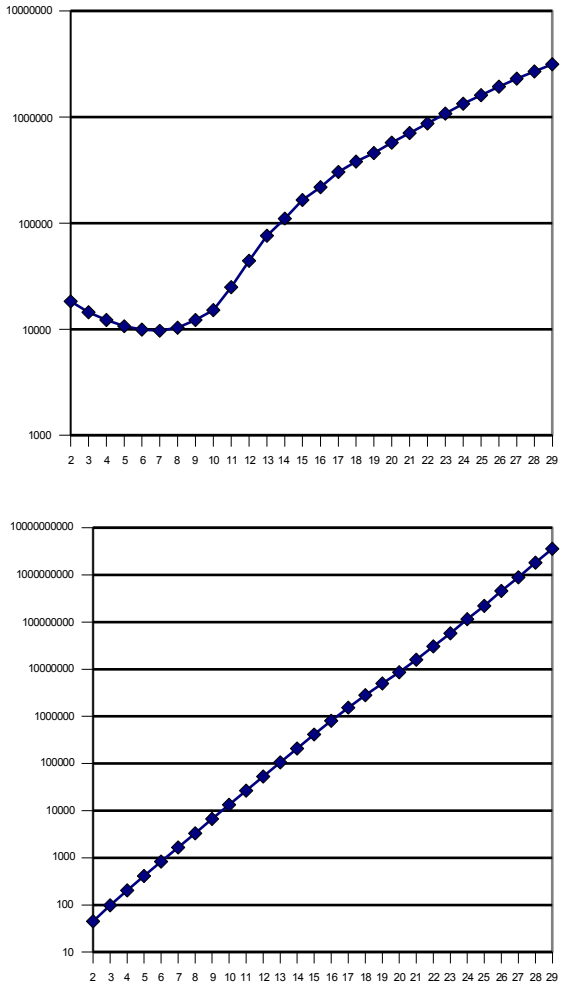

Figure 12: Logarithmic Step Count as Number of Elements Varies (Backtracking)

One important question with regards to run-time analysis now remains. If the partitioning problem is NP-Complete, it should have a stable exponentially increasing growth rate. Since the theory is that the ratio *m/n* is the control parameter for the partitioning problem, than holding that ratio constant while varying *m* and *n* together should produce this stable exponential growth rate. Figure 13 displays the step count for this experiment and Figure 14 the logarithmic version of the same data. Taken together they clearly show a stable exponential growth curve.

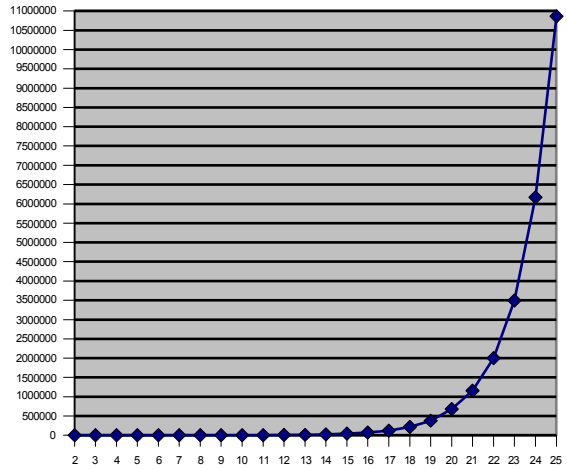

Figure 13: Step Count as m/n Ratio is Held Constant

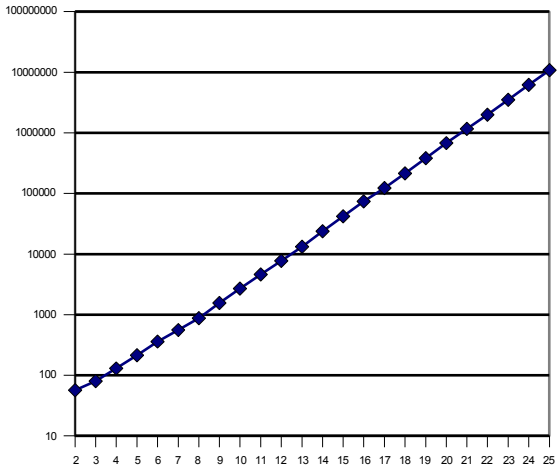

Figure 14: Logarithmic Step Count as m/n Ratio is Held Constant

5 Conclusion and Future Work

Based on the empirical analysis provided, several interesting observations can be made. While generally accepted as the quickest optimum solution provider for the partitioning problem, the "fast" Stearns algorithm suffers from performance problems when the average bit size of the integers in the problem instance is sufficiently larger than the number of integers in the problem instance. This points to an area for future improvement of the algorithm. It can also be said with empirical support that the *m/n* ratio appears to be the control parameter for locating the "harder" region of the partitioning problem.

A potential area of inquiry that should be addressed is how larger instances of the partitioning problem react to verify with empirical data that the conclusions and

observations made through this research hold up in larger problem instances. One of the major limitations in instance sizes for this research (and the reason the instance spaces tested is so small) is that the "fast" Stearns algorithm requires an array capable of holding every possible sum from the list's min value, to the sum of the entire list [8]. Since this requirement allows for direct array element access with no/few compares, it is integral the algorithm's speed it creates an artificial cap on the size of the instances that can be studied since array size is capped by the max int size of the language. One way to handle this in future research is to test to see if a block list or flip list for this array might be feasible without harming the algorithm. It might also be interesting to see if those changes would positively effect the Stearns algorithm in the large *m/n* value region.

References

[1] Cheeseman, Peter, Bob Kanefsky and William Taylor. "Where the Really Hard Problems Are." *Proceedings of IJCAI-91*, 331-337 1991.

[2] Hayes, Brian. "The Easiest Hard Problem." *Computing Science*, 90(2): 113-120 March-April 2002.

[3] Garey, M.R. and D.S. Johnson. *Computers and Intractability,* W.H. Freeman and Company, 1979.

[4] Gent, Ian and Toby Walsh. "Phase Transitions and Annealed Theories: Number Partitioning as a Case Study." *Proceedings of the 1996 European Conference on Artificial Intelligence*, 170-174 1996.

[5] Borgs, Christian, Jennifer Chayes and Boris Pittel. "Phase Transition and Finite-Size Scaling for the Integer Partitioning Problem." *Random Structures and Algorithms*, 19: 247-288 2001.

[6] Borgs, Christian, Jennifer Chayes and Boris Pittel. "Sharp Threshold and Scaling Window for the Integer Partitioning Problem." *Proceedings of the 2001 ACM Symposium on the Theory of Computing*, 330-336 2001.

[7] Mertens, Stephan. "A Physicist's Approach to Number Partitioning." *Theoretical Computer Science*, 265: 79-108 2001.

[8] Stearns, R.E. and H. B. Hunt III. "Power Indices and Easier Hard Problems." *Mathematical Systems Theory*, 23, 209-225 1990.

[9] Mertens, Stephan. "Random Costs in Combinatorial Optimization." *The American Physical Society*, 84(6): 1347-1350 February 2000.