Objective Functions for Multi-Way Number Partitioning

Richard E. Korf
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
korf@cs.ucla.edu

Abstract

The number partitioning problem is to divide a set of integers into a collection of subsets, so that the sum of the numbers in each subset are as nearly equal as possible. There are at least three natural objective functions for number partitioning. One is to minimize the largest subset sum, another is to maximize the smallest subset sum, and the third is to minimize the difference between the largest and smallest subset sums. I show that contrary to my previous claims, no two of these objective functions are equivalent for partitioning numbers three or more ways. Minimizing the largest subset sum or maximizing the smallest subset sum correspond to different practical applications of number partitioning, and both allow a recursive strategy for finding optimal solutions that is very effective in practice. Finally, a completely new version of this recursive strategy appears to reduce the asymptotic complexity of the algorithm, and results in orders of magnitude improvement over the best previous results for multi-way partitioning.

Introduction: Number Partitioning

Given a multiset of integers, the number partitioning problem is to divide them into a collection of mutually exclusive and collectively exhaustive subsets, so that the sum of the numbers in each subset are as nearly equal as possible. For example, if we divide the integers {13,9,9,6,6,6} into the subsets {13,6,6} and {9,9,6}, the sum of the numbers in the subsets is 25 and 24, respectively, which is optimal in this case. This is also a *perfect partition*. If the sum of all the numbers is divisible by the number of subsets, in a perfect partition all subsets will have the same sum. Otherwise they will differ by at most one. I focus here on optimal solutions.

Number partitioning is perhaps the simplest NP-complete problem to describe. It represents a very simple scheduling problem, often referred to as "multi-processor scheduling"(Garey and Johnson 1979). Given a set of jobs, each with an associated completion time, and two or more identical processors, assign each job to a processor to minimize the total time to complete all the jobs, assuming that each job must run on a single processor without interruption.

A recent application of number partitioning is to voting manipulation (Walsh 2009). Assume an election with three or more candidates, where each voter vetoes one of the candidates, and different voters have different weights. The winner is the candidate with the smallest total weight of vetoes. The question is how can a subgroup of the voters, known as the manipulators, guarantee the election of their chosen candidate, assuming that they know the votes of all the non-manipulators. The best strategy for the manipulators is to distribute the weights of their votes among the other candidates, so that the minimum total weight of vetoes received by the other candidates is greater than the weight of vetoes received by their chosen candidate from the non-manipulators. This is also a number partitioning problem.

Number Partitioning Objective Functions

There are at least three natural objective functions for number partitioning: minimizing the largest subset sum, maximizing the smallest subset sum, and minimizing the difference between the largest and smallest subset sums.

For two-way partitioning, all three of these objective functions are equivalent. The sum of the two subset sums in any two-way partition always equals the sum of all the numbers. Thus, given a two-way partition, reducing the largest subset sum must increase the smallest subset sum, and also reduce the difference between them. Similarly, increasing the smallest subset sum must reduce the largest subset sum, and also reduce the difference between them. Finally, reducing the difference between the largest and smallest subset sums requires increasing the smallest subset sum and decreasing the largest subset sum. Thus, optimizing any one of these objectives simultaneously optimizes the other two.

For three or more way partitioning, I previously assumed and reported that minimizing the largest subset sum was equivalent to minimizing the difference between the largest and smallest subset sum. For example, (Korf 1998) minimizes the difference between the largest and smallest subset sums, and states that, "This also minimizes the largest subset sum." (pp. 196). (Korf 2009) makes the same assumption.

For partitioning three or more ways, however, all three objective functions are distinct. For example, consider partitioning {13,9,9,6,6,6} three ways. Let partition x be {13,6}, {9,6}, and {9,6}, with subset sums of 19 and 15. Partition x maximizes the smallest subset sum (15). Let partition y be {13}, {9,9} and {6,6,6}, with subset sums of 13 and 18. Partition y minimizes the largest subset sum (18), even though

its minimum subset sum (13) is less than that of partition x (15). Thus, minimizing the largest subset sum is not equivalent to maximizing the smallest subset sum. Partition x also minimizes the difference between the largest and smallest subset sums ($19 - 15 = 4$). Since the subset sum difference of partition y is greater ($18 - 13 = 5$), minimizing the difference between the largest and smallest subset sum is not equivalent to minimizing the largest subset sum.

Next, consider partitioning {14,9,9,6,6,6} three ways. Let partition x be {14,6}, {9,6}, and {9,6}, with subset sums of 20 and 15. Partition x maximizes the smallest subset sum (15). Let partition y be {14}, {9,9} and {6,6,6}, with subset sums of 14 and 18. Partition y minimizes the difference between the largest and smallest subset sums ($18 - 14 = 4$), even though its minimum subset sum (14), is less than that of partition x (15). This example shows that maximizing the smallest subset sum is not equivalent to minimizing the difference between the largest and smallest subset sums.

Thus, all three objective functions are different for three-way partitioning. We can extend either of these counterexamples to k-way partitioning by adding $k - 3$ elements of size 16 or 17. For example, for four-way partitioning we would add one element of size 16 or 17 to either example.

Minimizing the largest subset sum corresponds to the multi-processor scheduling application, since minimizing the largest subset sum minimizes the time to complete all jobs. Maximizing the smallest subset sum corresponds to the voting manipulation application. These two objective functions are completely analogous to each other, and hence any algorithm that optimizes one of them can be easily transformed to one that optimizes the other.

Recursive Multi-Way Partitioning

Another property of these two objective functions is that they allow us to recursively decompose the search for an optimal partition (Korf 2009). We iterate over all possible two-way partitions of all the numbers, and for each, we optimally partition the two subsets as necessary. If the objective is to minimize the largest subset sum, or to maximize the smallest subset sum, then if a given top-level partition is preserved by any optimal complete partition, then optimally partitioning the two subsets will result in an overall optimal partition. For example, if any optimal three-way partition includes a given subset, then optimally partitioning the remaining elements two ways will result in an optimal three-way partition.

This principle of optimality does not apply if the objective is to minimize the difference between the largest and smallest subset sums. For example, consider partitioning {13,13,9,9,6,6,6} four ways. The optimal partition is {13}, {13}, {9,9} and {6,6,6}, with subset sums of 13 and 18, and a partition difference of $18 - 13 = 5$. If we first choose the singleton set {13}, however, and then optimally partition the remaining elements {13,9,9,6,6,6} three ways using this same objective function, we get the partition {13}, {13,6}, {9,6}, and {9,6}, with subset sums of 13, 19, and 15, and a suboptimal overall partition difference of $19 - 13 = 6$.

My previous work on multi-way partitioning (Korf 2009) minimized the difference between the largest and smallest subset sums. I mistakenly assumed that this objective was equivalent to minimizing the largest subset sum, and that this recursive principle of optimality applied to minimizing the difference between the largest and smallest subset sums. Thus, some of the four or five-way partitions computed in those experiments may not be optimal.

The algorithm of (Korf 2009) generated subsets by searching an inclusion-exclusion binary tree. At each node one branch included the corresponding number in all descendent subsets, and the other excluded it. The CKK algorithm (Korf 1998) was used for two-way partitioning.

I modified the algorithm of (Korf 2009) to minimize the largest subset sum, thus guaranteeing optimal solutions with this objective function, and reran all the experiments. The modification had very little effect on the running time. The experimentally observed asymptotic complexity was about $n^{1.836}$ for thee-way partitioning, $n^{1.897}$ for four-way partitioning, and $n^{2.142}$ for five-way partitioning of n elements.

I also implemented a new recursive algorithm for multi-way partitioning. Rather than using an inclusion-exclusion tree and CKK, I started with an algorithm that finds subsets with sums closest to a given target value (Schroeppel and Shamir 1981). This algorithm was modified to generate all subsets with sums within a given range. Furthermore, a different strategy was used for partitioning an odd number of ways. For example, when partitioning five ways, the previous algorithm chose a first subset, and then recursively partitioned the remaining elements four ways. In the new algorithm, the elements are first partitioning into subsets with sums of approximately two-fifths and three-fifths of the total, and then the first set is partitioned two ways and the second set three ways. The new algorithm reduces the running time by orders of magnitude in practice, and appears to reduce the asymptotic time complexity of optimal multi-way partitioning. The experimentally observed complexity was about $n^{1.418}$ for three-way partitioning, $n^{1.510}$ for four-way partitioning, and $n^{1.550}$ for five-way partitioning, compared to $n^{1.836}$, $n^{1.897}$, and $n^{2.142}$ for the previous algorithm.

Acknowledgments

This work was supported by NSF grant IIS-0713178. Thanks to Satish Gupta and IBM for the machine used to run the experiments.

References

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY: W. H. Freeman.

Korf, R. E. 1998. A complete anytime algorithm for number partitioning. *Artificial Intelligence* 106(2):181–203.

Korf, R. E. 2009. Multi-way number partitioning. In *Proceedings of IJCAI-09*, 538–543.

Schroeppel, R., and Shamir, A. 1981. A $t = o(2^{n/2}), s = o(2^{n/4})$ algorithm for certain np-complete problems. *SIAM Journal of Computing* 10(3):456–464.

Walsh, T. 2009. Where are the really hard manipulation problems? the phase transition in manipulating the veto rule. In *Proceedings of IJCAI-09*, 324–329.