Convergent Multi-Way Number Partitioning

David R. Stringer
self-published

Abstract

k-way number partitioning, when k is large, has solutions that are either non-optimal or impractical due to the use of decision trees that grow exponentially. These solutions focus on the process of partitioning the numbers into subsets. The new solution developed here starts with an established fast non-optimal algorithm but then focusses on improving the resulting subsets. This convergent algorithm is significantly faster than established optimal solutions yet early test results show it to be as good.

Introduction

In general, multi-partitioning seeks to divide a set of integers into k subsets so that the difference between the sum of numbers in any subset and any other subset is as small as possible. Partitioning into two subsets is relatively simple and various algorithms exist for this task. Some of these algorithms produce results very quickly but are non-optimal while others produce optimal results but take significantly more processing time.

Partitioning into k subsets when k is a significant fraction of the number count n (e.g. k = n/4), can be achieved by extending the k=2 algorithms. However, the processing time for optimal solutions with large k becomes impractical, typically on the order of hours. A more efficient algorithm is needed for large k.

Existing Algorithms

Existing algorithms are well-described in other articles so will only be listed here and briefly commented upon.

- Greedy Algorithm (GA)
- Complete Greedy Algorithm (CGA)
- Karmarker-Karp (KK)
- Complete Karmarker-Karp (CKK)
- Sequential Number Partitioning (SNP)
- Recursive Number Partitioning (RNP)

The GA and KK algorithms are non-optimal. CGA and CKK algorithms are optimal but achieve this by effectively searching decision trees that grow exponentially. SNP and RNP approach the solution piecewise, combining KK, CKK and decision trees to divide the numbers in smaller steps. Pruning of decision trees reduces processing time but not enough when k is large.

The problem of Large k

Previous work on multi-way partitioning has tended to assume small k, typically less than k=6. While it is possible to use SNP and RNP for large k, the processing time again becomes impractical as k increases.

It was noticed that one thing all the existing algorithms have in common is that they focus on the process of dividing the original n numbers into subsets. When k is large, the opportunity arises to change the focus from the original n numbers to the k non-optimal subsets. The problem then changes from partitioning to optimization of existing partitions. The aim was to find an algorithm that would take the k subsets resulting from a k-way KK algorithm and manipulate the subsets so as to converge on an optimal solution.

A Convergent Solution

First note that if the average subset sum (t/k where t is the sum of n numbers) is non-integer, a solution of zero difference between subset sums is impossible. An ideal solution therefore either has a subset-sum difference of one or, if t/k is integer, zero.

First run a fast non-optimal k-way multi-partitioning algorithm such as k-way KK to obtain k subsets. If the subset sum difference is ideal, as defined above, the solution has been found. No further processing is required. Otherwise, manipulate the subsets in a way that will converge on an optimal solution. The convergent algorithm works as follows.

First calculate the skewed mean of all the subset sums. This becomes the target subset sum that all subsets should converge upon. Then note, for each subset, its difference from this ideal: S = (subset sum – skewed mean) which may be positive, zero or negative. The aim is to minimize the largest absolute(S) of all subsets.

Convergence occurs by swapping numbers between any two subsets where such a swap will reduce the largest absolute(S) of both subsets. For example, if subset x has an S of +2 (referred to as Sx = +2) and subset y has Sy = -1, taking a 15 from x and swapping it with a 14 from y will yield Sx = +1 and Sy = 0. The largest absolute(S) is reduced from 2 to 1.

Swaps are found in a “converging pass” in which every subset pair is assessed to find a number swap that converges the pair. Swapping is not limited to single numbers. Swapping a 7 and an 8 from x with a 4 and a 10 from y would have achieved the same convergence as swapping a 15 with a 14. So, in each converging pass, convergence is achieved by testing all possible swaps between two subsets and, where more than one is possible, choosing the best. The best is the swap that minimizes the largest absolute(S) of both subsets. The maximum number count that any subset might contribute to any swap is b-1 where b is the cardinality of the subset.

The solution, as presented so far, only considers swaps between any two subsets. An optimal solution would need to consider multiple swaps between any combination of subsets. For example, subset x may swap numbers with y which swaps different numbers with z which swaps other numbers with x such that the 3-way swap reduces the largest absolute(S) of x, y and z. To avoid the complexity of 3-way up to k-way swaps, the concept of “number juggling” was introduced, as follows.

Whenever a converging pass through the subsets finds no possible swap and yet the largest absolute(S) is still non-ideal, the algorithm switches to a “juggling pass” which seeks swaps that are non-converging but also non-diverging. A “juggle swap” swaps numbers between any two subsets such that their largest absolute(S) is not increased. For example, if subset x swaps a 7 and an 8 for a 10 and a 5 of subset y, their sums and therefore S is not changed. But such a swap may expose a subsequent converging swap between x or y and another subset z that did not exist before. A juggle swap need not necessarily keep the same subset sum of each swapping subset. For example, if Sx = +3 and Sy = -1, a swap might be possible that yields Sx = +1 and Sy = -3. The juggle criterion is only that the largest absolute(S) of both subsets does not increase. After a successful juggling pass, the process returns to a converging pass.

Finally, in many cases, an ideal partition does not exist. There must therefore be a strategy for deciding when to give up the search for better convergence. The strategy chosen was to limit the number of times that a juggling pass can be followed by an unsuccessful converging pass. Also, the number of failed juggling passes allowed is of course only one.

Results

An implementation of the convergent algorithm can be seen in operation at https://www.eobar.org as the golf handicap balancer. Here, the algorithm takes the handicaps of players entering a golf tournament and partitions them so that teams are evenly balanced in their combination of player handicaps. In other words, the sum of handicaps in each team are as near equal to other teams as possible. This is a common problem faced by golf competition organizers and is non-trivial.

Note that this problem differs from many others, not only in having large k, but also in having a fixed cardinality of the subsets. The constraint on cardinality (team or subset size) increases the complexity of the task and therefore increases processing time.

Initial testing of the handicap balancer found that it was of the order of hundreds of times faster than the fastest of the other existing algorithms, and yet the handicap balancer obtained optimal solutions. However, exhaustive testing is yet to be done.

Conclusions

A convergent multi-way number partitioning algorithm for large k has been introduced and has been demonstrated to work in practice on a real-world problem. It differs from other existing algorithms by focussing on improving the results of a fast non-optimal algorithm. Further work is needed to compare its optimality with that of a complete algorithm. It would also be interesting to compare its processing time if the initial partitioning was done by the simplest possible, therefore fastest, division of the numbers. This is just splitting them into n/k chunks, ignoring the number values.

david@eobar.org