WIKIPEDIA

# Largest differencing method

In computer science, the **largest differencing method** is an algorithm for solving the partition problem and the multiway number partitioning. It is also called the **Karmarkar–Karp algorithm** after its inventors, Narendra Karmarkar and Richard M. Karp.[1] It is often abbreviated as **LDM.**[2] [3]

## Contents



## The algorithm

The input to the algorithm is a set *S* of numbers, and a parameter *k*. The required output is a partition of *S* into *k* subsets, such that the sums in the subsets are as nearly equal as possible. The main steps of the algorithm are:

1. Sort the numbers in descending order.
2. Repeatedly replace numbers by their difference, until one number remains.
3. Using backtracking, compute the partition.

### Two-way partitioning

For *k*=2, the main step (2) works as follows.

- Take the two largest numbers in *S*, remove them from *S*, and insert their difference (this represents a decision to put each of these numbers in a different subset).
- Proceed in this way until a single number remains. This single number is the difference in sums between the two subsets.

For example, if S = {8,7,6,5,4}, then the resulting difference-sets are 6,5,4,1, then 4,1,1, then 3,1 then 2.

Step 3 constructs the subsets in the partition by backtracking. The last step corresponds to {2},{}. Then 2 is replaced by 3 in one set and 1 in the other set: {3},{1}, then {4},{1,1}, then {4,7}, {1,8}, then {4,7,5}, {8,6}, where the sum-difference is indeed 2.

The runtime complextiy of this algorithm is dominated by the step 1 (sorting), which takes O(*n* log *n*).

Note that this partition is not optimal: in the partition {8,7}, {6,5,4} the sum-difference is 0. However, there is evidence that it provides a "good" partition:

- If the numbers are uniformly distributed in [0,1], then the expected difference between the two sums is $n^{-\Theta(\log(n)))}$. This also implies that the expected ratio between the maximum sum and the optimal maximum sum is $1+n^{-\Theta(\log(n)))}$ .[3]
- When there are at most 4 items, LDM returns the optimal partition.
- LDM always returns a partition in which the largest sum is at most 7/6 times the optimum.[4] This is tight when there are 5 or more items.[2]
- On random instances, this approximate algorithm performs much better than greedy number partitioning. However, it is still bad for instances where the numbers are exponential in the size of the set.[5]

## Multi-way partitioning

For any *k* ≥ 2, the algorithm can be generalized in the following way.[2]

- Initially, for each number *i* in *S*, construct a *k*-tuple of subsets, in which one subset is {*i*} and the other *k*-1 subsets are empty.
- In each iteration, select two *k*-tuples *A* and *B* in which the difference between the maximum and minimum sum is largest, and combine them in reverse order of sizes, i.e.: smallest subset in *A* with largest subset in *B*, second-smallest in *A* with second-largest in *B*, etc.
- Proceed in this way until a single partition remains.

Examples:

- If S = {8,7,6,5,4} and *k*=2, then the initial partitions are ({8},{}), ({7},{}), ({6},{}), ({5},{}), ({4},{}). After the first step we get ({6},{}), ({5},{}), ({4},{}), ({8},{7}). Then ({4},{}), ({8},{7}), ({6},{5}). Then ({4,7},{8}), ({6},{5}), and finally ({4,7,5},{8,6}), where the sum-difference is 2; this is the same partition as described above.
- If S = {8,7,6,5,4} and *k*=3, then the initial partitions are ({8},{},{}), ({7},{},{}), ({6},{},{}), ({5},{},{}), ({4},{},{}). After the first step we get ({8},{7},{}), ({6},{},{}), ({5},{},{}), ({4},{},{}). Then ({5},{},{}), ({4},{},{}), ({8},{7},{6}). Then ({5},{4},{}), ({8},{7},{6}), and finally ({5,6},{4,7},{8}), where the sum-difference is 3.
- If S = {5,5,5,4,4,3,3,1} and *k*=3, then after 7 iterations we get the partition ({5,5},{3,3,4},{1,4,5}).[2]

There is evidence for the good performance of LDM:[2]

- Simulation experiments show that, when the numbers are uniformly random in [0,1], LDM always performs better (i.e., produces a partition with a smaller largest sum) than greedy number partitioning. It performs better than the multifit algorithm when the number of items *n* is sufficiently large. When the numbers are uniformly random in [*o*, *o*+1], from some *o*>0, the performance of LDM remains stable, while the performance of multifit becomes worse as *o* increases. For *o*>0.2, LDM performs better.
- Let *f** be the optimal largest sum. If all numbers are larger than *f**/3, then LDM returns the optimal solution. Otherwise, LDM returns a solution in which the difference between largest and smallest sum is at most the largest number which is at most *f**/3.
- When there are at most *k*+2 items, LDM is optimal.

- When the number of items $n$ is between $k$+2 and 2$k$, the largest sum in the LDM partition is at most $\frac{4}{3} - \frac{1}{3(n-k-1)}$ times the optimum,
- In all cases, the largest sum in the LDM partition is at most $\frac{4}{3} - \frac{1}{3k}$ times the optimum, and there are instances in which it is at least $\frac{4}{3} - \frac{1}{3(k-1)}$ times the optimum.

### Balanced two-way partitioning

Benjamin Yakir[3] extended LDM to the *balanced* number partitioning problem, in which all subsets must have the same cardinality (up to 1). His BLDM algorithm for $k$=2 works as follows (where the items are ordered from largest to smallest);

- Replace numbers #1 and #2 by their difference; replace numbers #3 and #4 by their difference; etc.
- Once there are $n$/2 differences, run LDM.

## Implementation

The following Java code implements the first phase of Karmarkar–Karp. It uses a heap to efficiently find the pair of largest remaining numbers.

```
int karmarkarKarpPartition(int[] baseArr) {
    // create max heap
    PriorityQueue<Integer> heap = new PriorityQueue<Integer>(baseArr.length, REVERSE_INT_CMP);

    for (int value : baseArr) {
        heap.add(value);
    }

    while (heap.size() > 1) {
        int val1 = heap.poll();
        int val2 = heap.poll();
        heap.add(val1 - val2);
    }

    return heap.poll();
}
```

## An exact algorithm

The **complete Karmarkar–Karp algorithm (CKK)** finds an optimal solution by constructing a tree of degree $k!$.

- In the case $k$=2, each level corresponds to a pair of numbers, and the two branches correspond to taking their difference (i.e. putting them in different sets), or taking their sum (i.e. putting them in the same set).
- For general $k$, each level corresponds to a pair of $k$-tuples, and each of the $k!$ branches corresponds to a different way of combining the subsets in these tuples.

For $k$=2, CKK runs substantially faster than the Complete Greedy Algorithm (CGA) on random instances. This is due to two reasons: when an equal partition does not exist, CKK often allows more trimming than CGA; and when an equal partition does exist, CKK often finds it much faster and thus allows earlier termination. Korf reports that CKK can optimally partition 40 15-digit double-precision numbers in about 3 hours, while CGA requires about 9 hours. In practice, with $k$=2, problems of arbitrary size can be solved by CKK if the numbers have at most 12 significant digits; with $k$=3, at most 6 significant digits.[6]

CKK can also run as an anytime algorithm: it finds the KK solution first, and then finds progressively better solutions as time allows (possibly requiring exponential time to reach optimality, for the worst instances).[7]

## Previous mentions

An algorithm equivalent to the Karmarkar-Karp differencing heuristic is mentioned in ancient Jewish legal texts by Nachmanides and Joseph ibn Habib. The algorithm is used to combine different testimoines about the same loan.[8]

## References


1. Narendra Karmarkar and Richard M. Karp, "The differencing method of set partitioning", Tech report UCB/CSD 82/113, Computer science division, University of California, Berkeley, 1982
2. Michiels, Wil; Korst, Jan; Aarts, Emile (2003). "Performance ratios for the Karmarkar–Karp differencing method". *Electronic Notes in Discrete Mathematics*. **13**: 71–75. CiteSeerX 10.1.1.107.1332 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.107.1332). doi:10.1016/S1571-0653(04)00442-1 (https://doi.org/10.1016%2FS1571-0653%2804%2900442-1).
3. Yakir, Benjamin (1996-02-01). "The Differencing Algorithm LDM for Partitioning: A Proof of a Conjecture of Karmarkar and Karp" (https://pubsonline.informs.org/doi/abs/10.1287/moor.21.1.85). *Mathematics of Operations Research*. **21** (1): 85–99. doi:10.1287/moor.21.1.85 (https://doi.org/10.1287%2Fmoor.21.1.85). ISSN 0364-765X (https://www.worldcat.org/issn/0364-765X).
4. Fischetti, Matteo; Martello, Silvano (1987-02-01). "Worst-case analysis of the differencing method for the partition problem" (https://doi.org/10.1007/BF02591687). *Mathematical Programming*. **37** (1): 117–120. doi:10.1007/BF02591687 (https://doi.org/10.1007%2FBF02591687). ISSN 1436-4646 (https://www.worldcat.org/issn/1436-4646).
5. Hayes, Brian (March–April 2002), "The Easiest Hard Problem", *American Scientist*, Sigma Xi, The Scientific Research Society, vol. 90 no. 2, pp. 113–117, JSTOR 27857621 (https://www.jstor.org/stable/27857621)
6. Korf, Richard E. (1995-08-20). "From approximate to optimal solutions: a case study of number partitioning" (https://dl.acm.org/doi/abs/10.5555/1625855.1625890). *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1*. IJCAI'95. Montreal, Quebec, Canada: Morgan Kaufmann Publishers Inc.: 266–272. ISBN 978-1-55860-363-9.
7. Korf, Richard E. (1998-12-01). "A complete anytime algorithm for number partitioning" (http://www.sciencedirect.com/science/article/pii/S0004370298000861). *Artificial Intelligence*. **106** (2): 181–203. doi:10.1016/S0004-3702(98)00086-1 (https://doi.org/10.1016%2FS0004-3702%2898%2900086-1). ISSN 0004-3702 (https://www.worldcat.org/issn/0004-3702).
8. Ron Adin and Yuval Roichman (2015). "Combining witnesses: mathematical aspects" (https://u.cs.biu.ac.il/~tsaban/Pdf/AdinRoichman.pdf) (PDF). *BDD* (in Hebrew). Bar-Ilan University. **30**: –0.