Heuristic Algorithms for Balanced Multi-Way Number Partitioning

Jilian Zhang **Kyriakos Mouratidis** **HweeHwa Pang**
School of Information Systems
Singapore Management University
{jilian.z.2007, kyriakos, hhpang}@smu.edu.sg

Abstract

Balanced multi-way number partitioning (BMNP) seeks to split a collection of numbers into subsets with (roughly) the same cardinality and subset sum. The problem is NP-hard, and there are several exact and approximate algorithms for it. However, existing exact algorithms solve only the simpler, balanced two-way number partitioning variant, whereas the most effective approximate algorithm, BLDM, may produce widely varying subset sums. In this paper, we introduce the LRM algorithm that lowers the expected spread in subset sums to one third that of BLDM for uniformly distributed numbers and odd subset cardinalities. We also propose Meld, a novel strategy for skewed number distributions. A combination of LRM and Meld leads to a heuristic technique that consistently achieves a narrower spread of subset sums than BLDM.

1 Introduction

Number partitioning is a category of NP-complete problems that has been studied extensively. One of its variants is balanced multi-way number partitioning (BMNP). The input of BMNP is a set S of n numbers and a positive integer k; the output is a partition of S into subsets. The *subset sum* is the sum of the numbers in a subset, and the *subset cardinality* indicates how many numbers it contains. The objective in BMNP is to partition S into k subsets such that (i) the cardinality of each subset is either $\lfloor \frac{n}{k} \rfloor$ or $\lceil \frac{n}{k} \rceil$ numbers, and (ii) the *spread* (i.e., the difference) between the maximum and minimum subset sum is minimized.

BMNP has a wide range of applications, including multiprocessor scheduling [Dell'Amico and Martello, 2001], VLSI manufacturing [Tasi, 1995], etc. Consider, for example, a distributed system with k identical processors, and n tasks with different running-time requirements. Each processor has a fixed-size run queue for tasks. The scheduling of tasks to processors such that the CPU load is balanced and run queues do not exceed their limit is an instance of BMNP, where the task running-times play the role of numbers, processors the role of subsets, and run queue size that of subset cardinality.

BMNP is NP-hard [Dell'Amico and Martello, 2001]. To deal with its hardness, most existing approaches focus on suboptimal solutions (where the spread is higher than the minimum possible). Among them, the *balanced largest-first differencing method* (BLDM) is currently the most effective. Originally proposed for balanced 2-way partitioning in [Yakir, 1996], it was subsequently generalized to arbitrary k in [Michiels *et al.*, 2003]. BLDM first divides S into k-tuples (i.e., batches of k numbers). Then it selects and *folds* two of them, i.e., it couples their numbers and places the k produced pair sums into a new k-tuple, aiming to offset the variation within the original tuples. Folding continues iteratively until a single k-tuple remains; each of its elements corresponds to one of the k returned subsets.

While efficient, BLDM often produces partitions that are very far from optimal. We demonstrate that its spread is particularly high when the numbers in S follow a roughly uniform or a skewed distribution, and identify the reasons behind this. Motivated by these weaknesses, we propose heuristic algorithms LRM and Meld, each tailored to uniform and skewed data, respectively. We prove analytically and empirically that LRM reduces the expected spread to one third that of BLDM for uniform data. Meld, on the other hand, is shown to be significantly more effective than BLDM for non-uniform distributions (e.g., Zipf, normal, etc). Finally, we incorporate LRM, Meld and BLDM into a Hybrid algorithm that dynamically adapts to different data characteristics. Extensive experiments confirm that Hybrid consistently achieves lower spreads than BLDM, yet it remains practical in terms of computation time.

2 Background

2.1 Related Work

BMNP is a balanced variant of the number partitioning problem. The latter is NP-complete [Garey and Johnson, 1979], with many heuristic and exact algorithms proposed for it. Among them, the KK algorithm [Karmarkar and Karp, 1982] is the best approximate method for 2-way partitioning, generating a spread in subset sums in the order of $O(1/n^{\alpha \log n})$ for some constant α. [Korf, 1998] proposed CKK, an exact algorithm based on the principles of KK. CKK produces a global optimal solution for 2-way and multi-way partitioning, by exhaustively searching a binary tree that covers all possible combinations of subsets. The SNP and RNP algorithms [Korf, 2009] for k-way partitioning are more efficient

than CKK when $k \leq 5$. None of the above algorithms can ensure a balanced partitioning, i.e., equal subset cardinalities. Furthermore, the exact algorithms (CKK, SNP and RNP) are applicable only to small problem sizes.

BLDM is a modification of KK that guarantees the cardinality of the two resulting subsets to be $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ [Yakir, 1996]. There have been several attempts to extend BLDM from balanced 2-way to balanced multi-way number partitioning (BMNP). In [Michiels *et al.*, 2003], a generalized BLDM is proposed to perform balanced k-way partitioning for $k > 2$ in $O(n \log n)$ time. Hereafter, we refer to this generalized algorithm as BLDM.

2.2 Limitations of BLDM

As BLDM [Michiels *et al.*, 2003] is currently the most effective approximate algorithm for BMNP, we examine it in detail. Given a set of numbers S, BLDM performs balanced k-way partitioning as follows. First, S is padded with zero-value numbers, so that $n = |S| = bk$ for some integer b. The numbers in S are then sorted in descending order. The sorted sequence, denoted by $(v_1, v_2, ..., v_n)$, is split into b disjoint k-tuples $p_i = (v_{(i-1)k+1}, v_{(i-1)k+2}, ..., v_{ik})$, for $1 \leq i \leq b$. The spread in each p_i is $\delta(p_i) = v_{(i-1)k+1} - v_{ik}$. Next, BLDM repeatedly replaces the two k-tuples p_i and p_j with the largest spreads with a new k-tuple p'; p' is the result of *folding* p_i with p_j, by adding the first (largest) value in p_i with the last (smallest) value in p_j, the second (largest) value in p_i with the second last (smallest) in p_j, and so on. This process continues until a single k-tuple remains. Each of the k elements in this final tuple corresponds to one subset of the produced partitioning.

Although BLDM is very efficient, it performs poorly in two general scenarios. We explain each scenario with the aid of an example.

Example 1 *Consider a balanced* 4*-way partitioning on set* $S = \{12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1\}$. *BLDM divides* S *into three* 4*-tuples* $p_1 = (12, 11, 10, 9)$, $p_2 = (8, 7, 6, 5)$, *and* $p_3 = (4, 3, 2, 1)$. *Then, the two* 4*-tuples with largest spreads, say* p_1 *and* p_2 *(since all the* 4*-tuples have the same spread of* 3*), are folded to form a new* 4*-tuple* $p' = (17, 17, 17, 17)$. *Next,* p' *is folded with* p_3*, yielding the final* 4*-tuple* $(21, 20, 19, 18)$. *Tracing back the numbers that contributed to the final tuple, we derive the* 4*-way partitioning* $(\{12, 5, 4\}, \{11, 6, 3\}, \{10, 7, 2\}, \{9, 8, 1\})$*, with a spread in subset sums of* $21 - 18 = 3$*. This spread is as high as that of the original* 4*-tuples. In comparison, an optimal solution is* $(\{12, 5, 2\}, \{11, 8, 1\}, \{10, 7, 3\}, \{9, 6, 4\})$*, with a spread of only 1.*

The scenario in Example 1 occurs when the numbers in S follow a uniform (or roughly uniform) distribution and b is odd. In folding pairs of k-tuples, BLDM essentially offsets their spreads against each other. Since all the initial k-tuples have nearly the same spread (for uniform data), when b is odd BLDM will succeed in canceling out pairs of k-tuples as it is designed to do, leaving nothing to compensate for the last remaining k-tuple. Consequently, the reported partitioning inherits the spread of this last k-tuple. Figure 1 illustrates this situation in a uniform data scenario where $k = 4$ and $b = 3$.

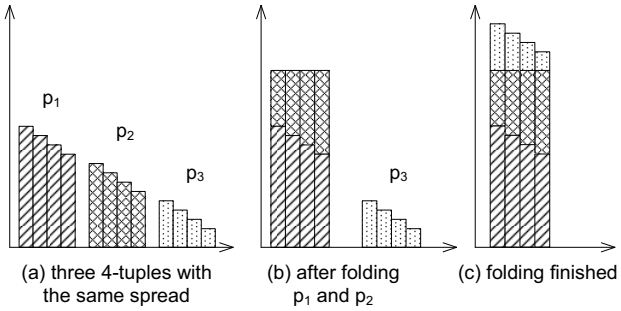


Figure 1: Folding process of BLDM on three 4-tuples

Each number is represented as a bar with height equal to its value. Folding p_1 and p_2 leaves no space for spread offsetting when p_3 is appended to derive the final k-tuple.

Example 2 *Consider a balanced* 4*-way partitioning on set* $S = \{60, 52, 40, 26, 19, 16, 14, 12, 10, 9, 6, 5\}$. *After dividing* S *into three* 4*-tuples* $p_1 = (60, 52, 40, 26)$, $p_2 = (19, 16, 14, 12)$, *and* $p_3 = (10, 9, 6, 5)$, *BLDM folds* p_1 *and* p_2 *first since they have the largest spreads of* 34 *and* 7. *The resulting* 4*-tuple* $p' = (72, 66, 56, 45)$ *is then folded with* p_3*, yielding the final* 4*-tuple* $(77, 72, 65, 55)$. *Thus, the reported partitioning is* $(\{60, 12, 5\}, \{52, 14, 6\}, \{40, 16, 9\}, \{26, 19, 10\})$*, with a spread of* $77 - 55 = 22$. *In contrast, the optimal solution* $(\{60, 6, 5\}, \{52, 10, 9\}, \{40, 14, 12\}, \{26, 19, 16\})$ *has a spread of only 10.*

The scenario illustrated in Example 2 occurs when the numbers in S follow a skewed distribution. The poor performance of BLDM stems from the k-tuple boundaries imposed right at the beginning of the process, forcing each subset sum in the final solution to be derived from exactly one number in each initial k-tuple. If the spread δ of some k-tuple exceeds the sum of spreads, Σ, of all the other k-tuples, BLDM will not obtain a final k-tuple with a spread smaller than $\delta - \Sigma$.

The shortcomings of BLDM highlighted above need to be addressed, owing to the abundance of uniformly distributed and of highly skewed data in real applications.

3 The LRM algorithm

In this section, we introduce an algorithm motivated by the first scenario where BLDM works poorly, i.e., uniformly distributed numbers with odd subset cardinality b. We begin with the case of $b = 3$ before extending to larger odd values of b.

The rationale of LRM is as follows. Let there be three k-tuples p_1, p_2 and p_3 with means of μ_1, μ_2 and μ_3, respectively. If folding the initial k-tuples could achieve a perfect balanced partitioning, each subset sum in the final k-tuple would be equal to $\mu_1 + \mu_2 + \mu_3$. Targeted at this ideal subset sum, we design our algorithm to fold the three tuples simultaneously (instead of performing two consecutive pairwise folds). Specifically, we optimistically form each subset from the leftmost[1] (L) number v_L in one tuple, the rightmost

[1]Since each k-tuple is sorted in descending order, its leftmost number has the maximum positive offset from the mean, whereas

(R) number v_R in another tuple, and a compensating number somewhere in the middle (M) of the remaining tuple that is closest to $\sum_{i=1}^{i=3} \mu_i - v_L - v_R$. This gives rise to LRM.

In forming a subset sum, LRM always performs the M operation (to pop a compensating number) in the input tuple that currently has the smallest spread[2]. The L and R operations are carried out on the tuples presently having the largest and second largest spreads, respectively. This is meant to reduce the chance of picking a compensating number that might be more useful for a subsequent subset sum; since the compensating number comes from the tuple with the smallest spread, the impact of a suboptimal choice (of compensating number) is small because the tuple is likely to hold other numbers with a similar value. Note that the strategy of picking the leftmost and rightmost numbers from the tuples currently having the highest spreads is consistent with the largest-first differencing strategy in KK [Karmarkar and Karp, 1982] and BLDM [Michiels *et al.*, 2003]. Algorithm 1 describes the LRM method.

To illustrate LRM, we refer again to Example 1 where we have $p_1 = (12, 11, 10, 9)$, $p_2 = (8, 7, 6, 5)$, $p_3 = (4, 3, 2, 1)$, and $\mu_1 + \mu_2 + \mu_3 = 19.5$. At first, LRM arbitrarily picks p_1 and p_2 for the L and R operations, as all three tuples have the same spread of 3. Popping 12 from p_1 and 5 from p_2, it chooses the compensating number from p_3 to be 2, because it has the closest value to $19.5 - 12 - 5 = 2.5$. This produces the first subset $\{12, 5, 2\}$. Now the spread in the remainders $p_1 = (11, 10, 9)$, $p_2 = (8, 7, 6)$ and $p_3 = (4, 3, 1)$ becomes 2, 2 and 3, respectively, causing LRM to pick p_3 and p_1 for the L and R operations in the second round. With numbers 4 from p_3 and 9 from p_1, 6 is chosen as the compensating number in p_2 that is nearest to $19.5 - 4 - 9 = 6.5$, leading to the second subset $\{4, 9, 6\}$. Repeating this process, we get the third and fourth subsets, $\{3, 10, 7\}$ and $\{1, 11, 8\}$. The final partitioning of $(\{12, 5, 2\}, \{4, 9, 6\}, \{3, 10, 7\}, \{1, 11, 8\})$, with a spread of only 1, matches the optimal solution and is a major improvement over BLDM's spread of 3.

Proposition 1 *Consider three k-tuples p_1, p_2 and p_3, comprising numbers that follow a uniform distribution. By combining p_1, p_2 and p_3 simultaneously, LRM generates a k-tuple p' with an expected spread $\delta(p')$ that is one third of that generated by BLDM.*

Proof. Given that the numbers in S are uniformly distributed, the three input k-tuples have roughly the same spread of, say, δ. Moreover, the expected difference between successive numbers in p_1, p_2 and p_3 is $C = \frac{\delta}{k-1}$. In each round of LRM, let p_L, p_R and p_M denote the k-tuples that produce the leftmost (v_L), rightmost (v_R) and compensating (v_M) numbers. Let μ_L, μ_R and μ_M denote the respective mean of the tuples. The subset sum generated is:

$$v_L + v_R + v_M = \sum_{i=1}^{3} \mu_i + (v_L - \mu_L) + (v_R - \mu_R) + (v_M - \mu_M) \quad (1)$$

the rightmost number has the maximum negative offset.

[2]We say "currently" because as numbers are removed from the tuples, their spread is updated to reflect their remaining contents.

On uniform input, the LRM strategy removes numbers from the k-tuples according to the following rotating pattern:

Round	p_1	p_2	p_3
1	leftmost	rightmost	middle
2	rightmost	middle	leftmost
3	middle	leftmost	rightmost
4	2^{nd} leftmost	2^{nd} rightmost	middle
5	2^{nd} rightmost	middle	2^{nd} leftmost
	...		

Thus, the leftmost number of a tuple is always matched with the rightmost number of another tuple, the 2^{nd} leftmost number is matched with the 2^{nd} rightmost number, etc. So, we have $(v_L - \mu_L) \approx (\mu_R - v_R)$, and Formula (1) simplifies to:

$$v_L + v_R + v_M \approx \sum_{i=1}^{3} \mu_i + (v_M - \mu_M) \quad (2)$$

$v_M - \mu_M$ is small in the early rounds, but grows gradually as the numbers in the middle of the tuples are used up. In round $i \in [1, k]$, $|v_M - \mu_M| = (\lfloor \frac{i-1}{6} \rfloor + 0.5)C$ for even k, whereas $|v_M - \mu_M| = (\lceil \frac{i+3}{6} \rceil - 1)C$ for odd k. Therefore, the last two rounds produce subset sums that, respectively, fall below and above $\sum_{i=1}^{3} \mu_i$ by the widest margin, and the difference between those two subset sums determines the final spread of the partitioning solution. In fact, the final spread is twice the value $|v_M - \mu_M|$ of the final round k, so:

$$spread = \begin{cases} 2(\lfloor \frac{k-1}{6} \rfloor + 0.5)\frac{\delta}{k-1} & \text{for even } k \\ 2(\lceil \frac{k+3}{6} \rceil - 1)\frac{\delta}{k-1} & \text{for odd } k \end{cases} \quad (3)$$

Recall that BLDM yields a final spread of δ, so the spread ratio of LRM w.r.t. BLDM converges to 1:3 for large k. □

LRM extends easily to odd values of b larger than 3. Specifically, the three k-tuples with the largest spreads are combined through LRM into an interim k-tuple with a small expected spread. The interim and the remaining k-tuples are iteratively folded pairwise, in the manner of BLDM, to cancel out their spreads until we are left with a single tuple. LRM has a time complexity of $O(n \log n)$, as we need to keep the numbers in each k-tuple sorted so as to find compensating numbers in logarithmic time (see line 7 in Algorithm 1).

Algorithm 1: LRM

Input: k-tuples p_1, p_2 and p_3 with means μ_1, μ_2 and μ_3
Output: a final k-tuple

1 $Sum = \mu_1 + \mu_2 + \mu_3$;
2 $p' = \emptyset$;
3 **while** $|p'| < k$ **do**
4 let p_L, p_R, p_M be the input k-tuple with the largest, second largest, and smallest spread, respectively;
5 v_L = the leftmost number removed from p_L;
6 v_R = the rightmost number removed from p_R;
7 v_M = the compensating number removed from p_M that is closest to $(Sum - v_L - v_R)$;
8 $p' = p' \cup \{v_L + v_R + v_M\}$;
9 return p';

4 The Meld Algorithm

Our second algorithm is designed to handle skewed data, the second scenario in which BLDM falls short. In case of skewed data, some k-tuples (e.g., tuple p_1 in Example 2) have a particularly large spread that cannot be effectively canceled out by folding with the remaining, smaller-spread k-tuples.

For ease of presentation, we consider the case where we have three input k-tuples (i.e., $b = 3$) before generalizing to arbitrary b. Let the tuples be p_1, p_2 and p_3, and suppose that the spread in p_1 is larger than the spread of the other two combined, i.e., $\delta(p_1) > \delta(p_2) + \delta(p_3)$. Assuming that $\delta(p_2) > \delta(p_3)$, BLDM would fold p_1 with p_2, and then the interim tuple with p_3. This leads to a final spread that is no less than $\delta(p_1) - \delta(p_2) - \delta(p_3) > 0$.

To avoid the pitfall, we deviate from BLDM's principle of eliminating the spread *whenever* a pair of k-tuples is folded, because in certain occasions a large interim spread may be needed to counterbalance the excessive spread in another k-tuple. Specifically, we combine p_2 and p_3 into an interim tuple p' with a spread $\delta(p')$ that is larger than $\delta(p_2)+\delta(p_3)$ and as close to $\delta(p_1)$ as possible, so that the subsequent folding of p' with p_1 can cancel out their respective spreads.

Our Meld algorithm achieves this by "melding" p_2 and p_3, that is, by uniting $p_2 = (v_{2,1}, \ldots, v_{2,k})$ and $p_3 = (v_{3,1}, \ldots, v_{3,k})$ into a common sorted sequence $p_\cup = (v_1, v_2, \ldots, v_{2k-1}, v_{2k})$, and then pairing its numbers so that the produced k-tuple p' has values that are (almost) uniformly spaced in $\left[\mu_2 + \mu_3 - \frac{\delta(p_1)}{2}, \mu_2 + \mu_3 + \frac{\delta(p_1)}{2}\right]$. We first identify in $p_\cup$ two number pairs $\mathcal{A} = \{v_i, v_j\}$ and $\mathcal{B} = \{v_l, v_m\}$ that add up to give v_i+v_j and v_l+v_m at the two extreme ends of p', such that $(v_i + v_j) - (v_l + v_m) \approx \delta(p_1)$. The next two number pairs $\mathcal{A}' = \{v'_i, v'_j\}$ and $\mathcal{B}' = \{v'_l, v'_m\}$, intended for the second position from the left and right of p', are chosen to meet condition $(v'_i + v'_j) - (v'_l + v'_m) \approx \delta(p_1) - \delta^-$, where $\delta^- = \frac{2\delta(p_1)}{k-1}$ is the desired rate of decline among the numbers in p'. This process is repeated to complete p'.

In pairing the numbers in $p_\cup$ as explained above, one may suggest using an exhaustive search. However, with $O(k^2)$ possible pairs, it takes $O(k^4)$ time to compute the difference between any two of them. Recall that $n = bk$, so the time complexity is $O(n^4)$, making exhaustive search impractical. Instead, we adopt a heuristic search strategy.

As shown in Algorithm 2 (lines 4 and 5), in the first iteration we place the largest (v_1) and smallest (v_{2k}) numbers of $p_\cup$ into pairs $\mathcal{A}$ and $\mathcal{B}$, respectively. For the second number in $\mathcal{A}$, we scan from v_{2k-1} back to v_2. Simultaneously, we scan from v_2 to v_{2k-1} to complete $\mathcal{B}$. The scan stops as soon as we encounter v_i and v_{2k-i+1} such that $(v_1+v_{2k-i+1})-(v_{2k}+v_i) \geq \delta(p_1)$ for some $2 \leq i \leq 2k-1$; these numbers complete the pairs, i.e., $\mathcal{A} = \{v_1, v_{2k-i+1}\}$ and $\mathcal{B} = \{v_i, v_{2k}\}$. The four chosen numbers are removed from $p_\cup$, and we proceed to find the next two pairs $\mathcal{A}$ and $\mathcal{B}$ (see lines 4 to 14). The process is repeated until $p_\cup$ becomes empty or it is left with two numbers only (see lines 6 to 8). The complexity of Meld is $O(k^2)$, since in the worst case it takes $O(k^2)$ steps to process all the numbers in $p_\cup$.

To illustrate the algorithm, we apply Meld to Example 2,

Algorithm 2: Meld

Input: k-tuple p_1, p_2 and p_3 with $\delta(p_1) > \delta(p_2) + \delta(p_3)$
Output: a final k-tuple p'

1 $p_\cup = p_2 \cup p_3$, sort $p_\cup$ in descending order;
2 $\delta^- = \frac{2\delta(p_1)}{k-1}$; $p' = \emptyset$;
3 **while** $|p_\cup| > 0$ **do**
4 let v_1 be the largest number in $p_\cup$;
5 let $v_{|p_\cup|}$ be the smallest number in $p_\cup$;
6 **if** $|p_\cup| = 2$ **then**
7 $p' = p' \cup \{v_1 + v_{|p_\cup|}\}$;
8 break out of the **while** loop;
9 **for** $i = 2$ **to** $|p_\cup| - 1$ **do**
10 **if** $(v_1 + v_{|p_\cup|-i+1}) - (v_{|p_\cup|} + v_i) \geq \delta(p_1)$ **then**
11 break out of the **for** loop;
12 $p' = p' \cup \{v_1 + v_{|p_\cup|-i+1}\} \cup \{v_{|p_\cup|} + v_i\}$;
13 Remove $v_1, v_{|p_\cup|}, v_i, v_{|p_\cup|-i+1}$ from $p_\cup$;
14 $\delta(p_1) = \delta(p_1) - \delta^-$;
15 fold p_1 with p' and store the result in p';
16 return p';

where $S = \{60, 52, 40, 26, 19, 16, 14, 12, 10, 9, 6, 5\}$. Here $p_\cup = p_2 \cup p_3 = \{19, 16, 14, 12, 10, 9, 6, 5\}$, $\delta(p_1) = 60 - 26 = 34$ and $\delta^- = 34 * 2/3 = 22.7$. After initializing $\mathcal{A} = \{19, \_\}$ and $\mathcal{B} = \{5, \_\}$, Meld scans from 6 towards 16 to complete $\mathcal{A}$ and from 16 to 6 for $\mathcal{B}$. The scan produces numbers 16 and 6, without satisfying the condition in line 10 of Algorithm 2. The iteration is completed by adding $19+16$ and $6+5$ to p', and removing these four numbers from $p_\cup$. The next iteration starts with 14 in $\mathcal{A}$ and 9 in $\mathcal{B}$. The scan in lines 9 to 11 yields $\mathcal{A} = \{14, 12\}$ and $\mathcal{B} = \{10, 9\}$, again without passing the test in line 10. Now we have $p' = (19+16, 6+5, 14+12, 9+10)$=$(35, 26, 19, 11)$. Folding p_1 with p' produces the final partitioning $(60 + 11, 52 + 19, 40 + 26, 26 + 35) = (71, 71, 66, 61)$, with an optimal spread of 10, a 50% improvement over BLDM's spread of 22.

The following proposition shows formally that melding p_2 and p_3 indeed achieves the objective of a wider spread than folding.

Proposition 2 *Melding k-tuples p_2 and p_3 produces a k-tuple p' in which each element is the sum of two numbers in $p_2 \cup p_3$, such that $\delta(p') \geq \delta(p_2) + \delta(p_3)$.*

Proof. Since the numbers in every k-tuple are sorted in descending order by construction, we have:

$$\begin{aligned} \delta(p_2) + \delta(p_3) &= (v_{2,1} - v_{2,k}) + (v_{3,1} - v_{3,k}) \\ &= (v_{2,1} + v_{3,1}) - (v_{2,k} + v_{3,k}) \quad (4) \end{aligned}$$

Now consider the number pairs $\mathcal{A}$ and $\mathcal{B}$ found in the first iteration of Algorithm 2. If the scan in lines 9 to 11 completes without passing the test in line 10, $\mathcal{A}$ holds the two largest numbers v_1 and v_2 in $p_\cup$, with $v_1 + v_2 \geq v_{2,1} + v_{3,1}$. Moreover, $\mathcal{B}$ holds the two smallest numbers v_{2k-1} and v_{2k} in $p_\cup$, with $v_{2k-1} + v_{2k} \leq v_{2,k} + v_{3,k}$. Thus, $\delta(p') \geq \delta(p_2) + \delta(p_3)$. If the scan terminates early, then $\mathcal{A} = \{v_1, v_{2k-i+1}\}$ and

$\mathcal{B} = \{v_i, v_{2k}\}$ lead to $\delta(p') = (v_1 + v_{2k-i+1}) - (v_i + v_{2k})$ that is just above $\delta(p_1)$, so $\delta(p') > \delta(p_2) + \delta(p_3)$. □

Meld extends to situations where S is split into more than three k-tuples. We repeatedly test whether the three k-tuples p_i, p_j and p_l with the largest spreads satisfy the condition $\delta(p_i) > \delta(p_j) + \delta(p_l)$. If so, we perform melding to combine them into a new k-tuple; if not, we resort to a folding operation on p_i and p_j. Through this process of melding and folding operations, we eventually obtain a single k-tuple that represents the final partitioning.

5 The Hybrid Algorithm

LRM and Meld can be applied to produce high-quality solutions when the numbers to be partitioned are known (or have been tested) to follow a uniform or skewed distribution. To cope with arbitrary input data without a priori knowledge of their distribution, we incorporate LRM, Meld and the folding operation of BLDM into a Hybrid algorithm. Specifically, whenever the three k-tuples p_i, p_j and p_l with the largest spreads satisfy the condition $\delta(p_i) > \delta(p_j) + \delta(p_l)$, Meld is executed. Otherwise, we perform LRM or folding, depending on whether the number of remaining k-tuples is odd or even. The process is repeated until we are left with a single k-tuple.

6 Empirical Validation

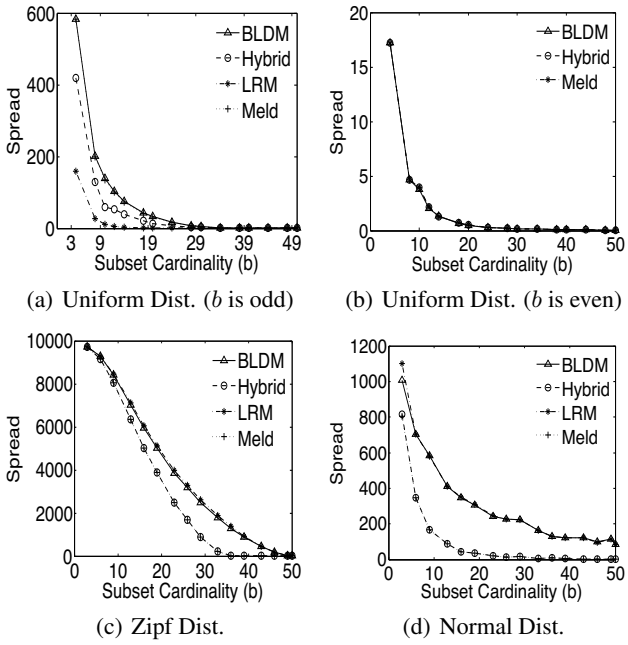
We implemented BLDM, LRM, Meld and Hybrid in C++ and evaluated them on a PC with an Intel Core Duo 2GHz CPU. We used three kinds of datasets: (a) The first contains numbers that are normally distributed with a mean of 1000 and variance from $0.1 \times mean$ to $1.0 \times mean$. (b) The second dataset is uniformly distributed, with numbers drawn from $[0, 2000]$. (c) The last dataset follows a Zipf distribution $f(x) = \frac{1}{x^\theta}$ in $[0, 10000]$, with θ from 0.1 to 2. Each reported measurement is the average over 100 trials.

6.1 Impact of Subset Cardinality (b)

In the first experiment, we measure the spread as we vary the subset cardinality b from 3 to 50 (because LRM and Meld are designed for $b \geq 3$), while fixing the number of subsets k to 500. This setting corresponds to a balanced 500-way partitioning. In tandem with the varying b, the dataset size n increases from 3×500 to 50×500.

For the uniform dataset, we report the results separately for odd and even b, because LRM is designed specifically for odd b settings. Figure 2(a) shows the spreads for odd b values, with $b = 3, 5, 7..., 49$ on the x-axis. The results confirm that LRM achieves $\frac{1}{3}$ the spread of BLDM, as proven in Proposition 1. Meld performs as poorly as BLDM, which is not surprising; since the condition for melding is never met for k-tuples with nearly equal spreads, Meld resorts to folding operations instead. Hybrid is slightly inferior to LRM, because the former may invoke LRM multiple times (instead of just once), which is not ideal for uniformly distributed data.

Figure 2(b) illustrates the spread for uniform data and even b values, with $b = 4, 6, 8..., 50$ on the x-axis. For even b, LRM degenerates to BLDM and is thus omitted from the chart. BLDM achieves partitioning solutions with very small spreads. This graceful performance is possible only because



(a) Uniform Dist. (b is odd) (b) Uniform Dist. (b is even)

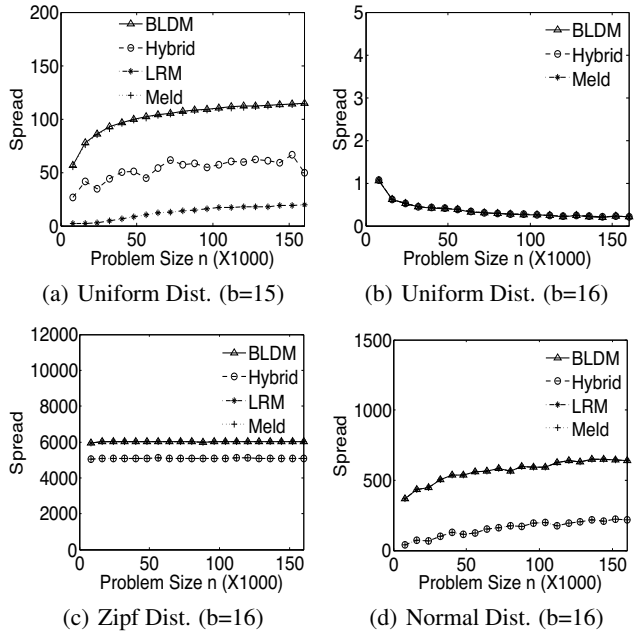
(c) Zipf Dist. (d) Normal Dist.

Figure 2: Performance comparison with k fixed at 500

in this experiment there is an even number of k-tuples with similar spreads that successfully offset each other. Since the execution conditions for LRM and melding are not met, Hybrid relies exclusively on folding operations, leading to the same partitioning solutions as BLDM. Likewise for Meld.

Next, we turn to the results obtained with the Zipf dataset, depicted in Figure 2(c). Due to space limitation, the figure only includes results for $\theta = 1.2$ (where about 95% of the numbers fall in 20% of the data space). Clearly, BLDM does not handle skewed data well, for the reasons discussed in Section 2.2. LRM exhibits similar performance to BLDM because, even when b is odd, the data skew in the Zipf distribution is too high to be offset by applying LRM only once on the first three k-tuples. In comparison, Meld handles skewed data successfully, especially for $b > 10$. We note that Meld works even better in (omitted) experiments with a larger θ. Hybrid behaves similarly to Meld, since it invokes the melding operation extensively.

Turning to normally distributed data, we plot results for a variance of $0.6 \times mean$ in Figure 2(d) (the relative performance of the algorithms is similar across all variances tested). Meld and Hybrid outperform BLDM vastly, producing partitioning solutions with spreads less than $\frac{1}{10}$ of those generated by BLDM for b between 18 and 50. This wide difference arises because the normal distribution leads to relatively few large and small numbers, while the majority of the values cluster around the mean. Thus, when the dataset S is carved into k-tuples, the first few have steep spreads, followed by many tuples with smaller and smaller spreads, before the pattern reverses. Meld and Hybrid are able to offset those k-tuples with large spreads in the early iterations, before switching to folding operations until termination. BLDM and LRM, however, are not adept at handling such data.



(a) Uniform Dist. (b=15) (b) Uniform Dist. (b=16)

(c) Zipf Dist. (b=16) (d) Normal Dist. (b=16)

Figure 3: Performance comparison with fixed b

6.2 Impact of the Problem Size (n)

The next experiment studies the effect of the dataset size, i.e., n, on the various algorithms. We increase n progressively from $8,000$ to $160,000$ while fixing the subset cardinality b. We report only results for $b = 16$ (as well as $b = 15$ for uniform data). Results for other b settings follow a similar trend to those shown here. Note that k increases in tandem with n, from 500 to $10,000$.

For $b = 15$ and uniform data (Figure 3(a)), LRM and Hybrid are consistently better than BLDM. In Figure 3(b), for $b = 16$ and uniform data, the spreads produced by all the algorithms drop and then remain close to zero as n increases, verifying that even cardinalities are generally easier to handle. For Zipf numbers, Figure 3(c) shows that Meld and Hybrid outperform BLDM consistently by about 17% across different n settings. For normally distributed data (Figure 3(d)), Meld and Hybrid achieve spreads that are about 10% to 34% those of BLDM. Across all experiments in Figure 3, the relative performance of LRM, Meld, Hybrid and BLDM is stable with respect to n, because it is the data distribution that has a larger effect on their effectiveness.

6.3 Computation Overhead

Finally, we consider the CPU overhead of the various algorithms. At $n = 25,000$ and $k = 500$, BLDM executes in under 80 ms in all of our experiments. Despite having the same time complexity $O(n \log n)$ as BLDM, the actual CPU time incurred by LRM is higher, at about 100 ms. In contrast, Meld and Hybrid are more computationally intensive, owing to the melding operation; their CPU times are less than 600 ms for the normal distribution, 780 ms for the Zipf data, and 200 ms for the uniform dataset. Nevertheless, LRM, Meld and Hybrid are all practical, having very reasonable execution times. It is worth mentioning that even for one million numbers (with, say, $k = 500$ and normal distribution), they all complete in less than 4 seconds.

7 Conclusion

In this paper, we study the balanced k-way number partitioning problem. Given a set of numbers, the goal is to partition it into k subsets of equal cardinalities so that the subset sums are as similar as possible. With the objective of designing practical solutions for large problem settings, we introduce two heuristic methods. The first, called LRM, utilizes a predictable pattern of adding numbers across three batches of k in order to produce k partial sums that are similar in magnitude. The second method, Meld, employs a (seemingly counterintuitive) strategy of melding two batches of k numbers into a batch of k widely varying partial sums, before offsetting them against another high-variance batch. To complete our framework, we combine LRM and Meld into a Hybrid algorithm that dynamically adapts to different data characteristics. Extensive experiments confirm the effectiveness of LRM and Meld for uniform and skewed data, respectively, while Hybrid consistently produces high-quality partitioning solutions within short execution times.

References

[Dell'Amico and Martello, 2001] M. Dell'Amico and S. Martello. Bounds for the cardinality constrainted $p||c_{max}$ problem. *Journal of Scheduling*, 4:123–138, 2001.

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

[Karmarkar and Karp, 1982] N. Karmarkar and R. Karp. The differencing method of set partitioning. Technical Report UCB/CSD 82/113, University of California, Berkeley, CA, 1982.

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

[Korf, 2009] Richard E. Korf. Multi-way number partitioning. In *International Joint Conference on Artificial Intelligence*, pages 538–543, 2009.

[Michiels *et al.*, 2003] Wil Michiels, Jan H. M. Korst, Emile H. L. Aarts, and Jan van Leeuwen. Performance ratios for the differencing method applied to the balanced number partitioning problem. In *Symposium on Theoretical Aspects of Computer Science*, pages 583–595, 2003.

[Tasi, 1995] L. Tasi. The modified differencing method for the set partitioning problem with cardinality constraints. *Discrete Applied Mathematics*, 63:175–180, 1995.

[Yakir, 1996] B. Yakir. The differencing algorithm LDM for partitioning: A proof of a conjecture of Karmarkar and Karp. *Mathematics of Operations Research*, 21(1):85–99, 1996.