The Partition Problem

Submitted to: Prof. Neelima Gupta

Department of Computer Science, University of Delhi

Submitted by:

Pankaj Kumar Chaudhary (Roll No. 34)

Priyanshu Gupta (Roll No. 39)

November 1, 2025

Table of Contents

Overview

- The **Partition Problem (PP)** is a classic NP-hard optimization problem.
- **Objective:** Minimize the difference of two subsets.
- **Applications:**
  - Load balancing and scheduling
  - Resource allocation
  - Cryptography and combinatorial optimization
- Decision version is NP-complete; optimization version is NP-hard.
- Classical references: Garey and Johnson (1979), Karp (1972).

Problem Statement

Formal Definition

Given a set $S = \{a_1, a_2, \ldots, a_n\}$, find disjoint subsets S_1, S_2 such that:

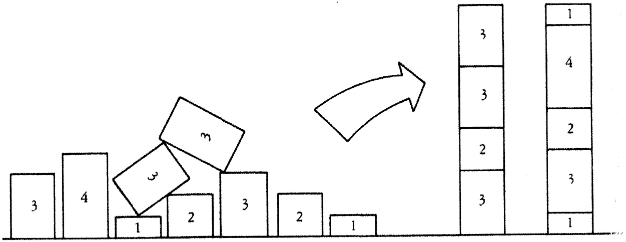
$$\sum_{a_i \in S_1} a_i = \sum_{a_j \in S_2} a_j$$

Optimization Objective

Minimize:

$$\Delta = \left| \sum_{a_i \in S_1} a_i - \sum_{a_j \in S_2} a_j \right|$$

Visualization of the Partition Problem



Interpretation

The visualization illustrates how the Partition Problem divides a set of weighted elements into two subsets such that their total sums are as balanced as possible. $\Delta = |\sum S_1 - \sum S_2|$.

Brute Force Algorithm: Concept

- The brute force algorithm checks **all possible partitions** of the set S.
- For each partition, compute the difference between subset sums:

$$\Delta = \left| \sum_{a_i \in S_1} a_i - \sum_{a_j \in S_2} a_j \right|$$

- Return the partition that minimizes Δ.
- Time complexity: $O(2^n)$, since each element can go in either S_1 or S_2.

Brute Force Algorithm (Pseudocode)

Algorithm 1: Brute Force Partition Algorithm

```
Input: Array A[1..n]
total ← sum(A);
best_diff ← ∞;
for each subset S of A do
    s ← sum(S);
    diff ← |total − 2 × s|;
    if diff < best_diff then
        best_diff ← diff;
        best_subset ← S;
    end
end
Output: best_subset, best_diff;
```

Complexity: $O(2^n \times n)$ time, $O(1)$ space.

Brute Force Example (n = 4)

Example Set: $S = \{3, 1, 4, 2\}$

All possible partitions (showing only a few):

- $S_1 = \{3, 1\}, S_2 = \{4, 2\} \rightarrow \Delta = |4 - 6| = 2$
- $S_1 = \{3, 4\}, S_2 = \{1, 2\} \rightarrow \Delta = |7 - 3| = 4$
- $S_1 = \{3, 2\}, S_2 = \{4, 1\} \rightarrow \Delta = |5 - 5| = 0$

Optimal Partition:
$S_1 = \{3, 2\}, S_2 = \{4, 1\} \rightarrow \Delta_{min} = 0$

Set $S = \{3, 1, 4, 2\}$.

Try all partitions:

S_1	S_2	Δ
$\{3,1\}$	$\{4,2\}$	2
$\{3,4\}$	$\{1,2\}$	4
$\{3,2\}$	$\{4,1\}$	0

Best

Figure: Example of Brute Force Partition Enumeration

Greedy Approximation Algorithm for Partition Problem

Input : Set of numbers $S = \{a_1, a_2, \ldots, a_n\}$
Output: Two subsets S_1 and S_2 minimizing difference Δ

1. Sort all elements of S in **non-increasing order**.

2. Initialize two empty subsets $S_1 \leftarrow \emptyset$, $S_2 \leftarrow \emptyset$ and their sums $L_1 \leftarrow 0$, $L_2 \leftarrow 0$.

3. For each element a_i in sorted order:

1. **if** $L_1 \leq L_2$ **then**
   | Assign a_i to S_1; update $L_1 \leftarrow L_1 + a_i$
   end
2. **else**
   | Assign a_i to S_2; update $L_2 \leftarrow L_2 + a_i$
   end

4. Compute the absolute difference:

$$\Delta = |L_1 - L_2|$$

5. Return subsets S_1, S_2, and difference Δ.

Greedy Approximation: Example

Example: $S = \{7, 6, 5, 3, 2\}$

Sorted in decreasing order: $[7, 6, 5, 3, 2]$

Step-by-step assignment:

- Start: $S_1 = \emptyset, L_1 = 0;\ S_2 = \emptyset, L_2 = 0$
- Assign $7 \to S_1 \to L_1 = 7, L_2 = 0$
- Assign $6 \to S_2 \to L_1 = 7, L_2 = 6$
- Assign $5 \to S_2 \to L_1 = 7, L_2 = 11$
- Assign $3 \to S_1 \to L_1 = 10, L_2 = 11$
- Assign $2 \to S_1 \to L_1 = 12, L_2 = 11$

Final Partition:
$S_1 = \{7, 3, 2\}, \quad S_2 = \{6, 5\}$

Difference: $\Delta = |L_1 - L_2| = |12 - 11| = 1$

Karmarkar–Karp Heuristic: Concept

- The **Karmarkar–Karp Heuristic (KK)** is a fast and effective method for the Partition Problem.
- It repeatedly selects the two largest elements a and b from the set.
- Replace them with their absolute difference $|a - b|$.
- Continue until only one number remains — this represents the final imbalance (difference between subset sums).
- **Time Complexity:** $O(n \log n)$ using a max-heap.
- Produces solutions close to optimal in practice, though not guaranteed optimal.

Karmarkar–Karp Heuristic: Example

Example: Consider the set $S = \{8, 7, 6, 5, 4\}$

1. Pick two largest numbers: $8, 7 \Rightarrow |8 - 7| = 1$
2. New set: $\{6, 5, 4, 1\}$
3. Pick two largest numbers: $6, 5 \Rightarrow |6 - 5| = 1$
4. New set: $\{4, 1, 1\}$
5. Pick two largest numbers: $4, 1 \Rightarrow |4 - 1| = 3$
6. New set: $\{3, 1\}$
7. Pick two largest numbers: $3, 1 \Rightarrow |3 - 1| = 2$
8. Remaining number: 2

Result:

$$\Delta = 2$$

Interpretation

The remaining number (2) represents the difference between the two subset sums. The algorithm gives a near-optimal partition with minimal imbalance.

Experimental Setup: Overview

- Goal: Evaluate and compare three algorithms for the **Partition Problem**.
- Algorithms studied:
  - **Brute Force (Exact):** Explores all possible partitions ($O(2^n)$).
  - **Greedy LPT Approximation:** Sorts and assigns elements to the subset with minimum current sum ($O(n \log n)$).
  - **Karmarkar–Karp Heuristic:** Repeatedly replaces two largest numbers by their difference ($O(n \log n)$).
- Implementation: Python 3.11

Dataset and Generation

- A custom dataset of 24 positive integers was used to simulate load balancing.
- Values were chosen in the range $[1, 100]$ to include small and large weights.
- All three algorithms executed on each subset size, results recorded.

Comparison 1: Approximation and Heuristic vs Brute Force

- The Brute Force algorithm was used as the **optimal baseline**.
- Both **Greedy LPT** and **Karmarkar–Karp** were evaluated against it.
- Theoretical Approximation Bounds:

$$\rho_{\text{Greedy}} \leq \frac{4}{3} - \frac{1}{3m} = 1.167, \quad \rho_{\text{KK}} \approx 1.25$$

Experimental Results Table

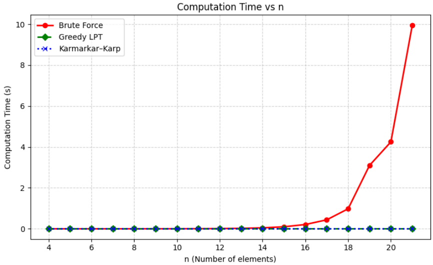
figureComparison 1 — Brute Force vs Greedy LPT and Karmarkar–Karp

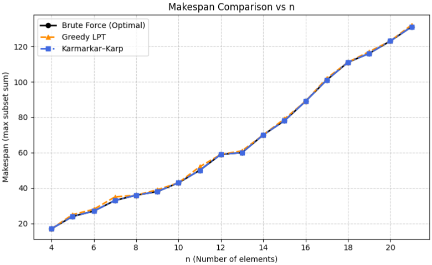
=== Comparison 1: Makespan-based Approximation Factors ===

n	BF_diff	Greedy_diff	KK_diff	BF_makespan	Greedy_makespan	KK_makespan	BF_time	Greedy_time	KK_time	Greedy_factor_emp	KK_factor_emp
4	1	1	1	17	17	17.0	0.00004	0.00001	0.00001	1.00000	1.0
5	2	4	2	24	25	24.0	0.00005	0.00000	0.00001	1.04167	1.0
6	0	2	0	27	28	27.0	0.00009	0.00000	0.00000	1.03704	1.0
7	1	5	1	33	35	33.0	0.00025	0.00000	0.00001	1.06061	1.0
8	1	1	1	36	36	36.0	0.00056	0.00001	0.00001	1.00000	1.0
9	1	3	1	38	39	38.0	0.00102	0.00000	0.00001	1.02632	1.0
10	1	1	1	43	43	43.0	0.00341	0.00001	0.00001	1.00000	1.0
11	0	4	0	50	52	50.0	0.00550	0.00001	0.00002	1.04000	1.0
12	1	1	1	59	59	59.0	0.01756	0.00002	0.00002	1.00000	1.0
13	0	2	0	60	61	60.0	0.03392	0.00001	0.00002	1.01667	1.0
14	0	0	0	70	70	70.0	0.07654	0.00002	0.00002	1.00000	1.0
15	0	2	0	78	79	78.0	0.18743	0.00001	0.00003	1.01282	1.0
16	0	0	0	89	89	89.0	0.30700	0.00001	0.00002	1.00000	1.0
17	1	3	1	101	102	101.0	0.65469	0.00001	0.00004	1.00990	1.0
18	0	0	0	111	111	111.0	1.78729	0.00002	0.00003	1.00000	1.0
19	0	2	0	116	117	116.0	3.97585	0.00002	0.00003	1.00862	1.0
20	0	0	0	123	123	123.0	8.19210	0.00002	0.00003	1.00000	1.0

- Brute Force grows exponentially and becomes infeasible beyond $n = 20$.
- Greedy LPT and Karmarkar–Karp both achieve near-optimal results with negligible runtime.

Experimental Graphs





Approximation Factor Summary

```
=================== Approximation Table (Makespan-based) ====================
n     BF_ms      Greedy_ms    Greedy_Factor    KK_ms      KK_Factor
---------------------------------------------------------------------
4     17         17           1.0000           17         1.0000
5     24         25           1.0417           24         1.0000
6     27         28           1.0370           27         1.0000
7     33         35           1.0606           33         1.0000
8     36         36           1.0000           36         1.0000
9     38         39           1.0263           38         1.0000
10    43         43           1.0000           43         1.0000
11    50         52           1.0400           50         1.0000
12    59         59           1.0000           59         1.0000
13    60         61           1.0167           60         1.0000
14    70         70           1.0000           70         1.0000
15    78         79           1.0128           78         1.0000
16    89         89           1.0000           89         1.0000
17    101        102          1.0099           101        1.0000
18    111        111          1.0000           111        1.0000
19    116        117          1.0086           116        1.0000
20    123        123          1.0000           123        1.0000
=====================================================================
```

Comparison 2: Approximation vs Heuristic

- This experiment evaluates **Greedy LPT (Approximation)** and **Karmarkar–Karp (Heuristic)** on large datasets.
- **Dataset sizes:** $n = \{100, 200, 300, 400, 500, 600\}$.
- Each dataset consists of random integers
- Brute Force is excluded since it becomes infeasible for large inputs.

Experimental Results Table (Large Datasets)

figureComparison 2 — Greedy LPT vs Karmarkar–Karp (Large Datasets)

n	Greedy_diff	KK_diff	Greedy_makespan	KK_makespan	Greedy_time	KK_time
100	7	1	23555	23552.0	0.00007	0.00018
200	1	1	53143	53143.0	0.00013	0.00024
300	7	1	76894	76891.0	0.00011	0.00037
400	4	0	96614	96612.0	0.00015	0.00049
500	1	1	117922	117922.0	0.00025	0.00060
600	1	1	152394	152394.0	0.00025	0.00077

- The table summarizes makespan and execution time for each dataset size.

Experimental Graphs (Comparison 2)

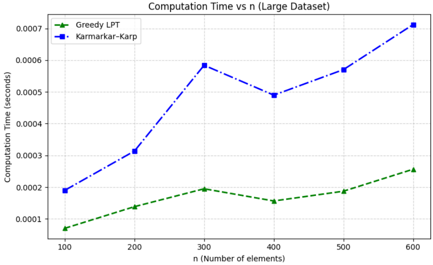


Figure: *

(a) Computation Time vs n

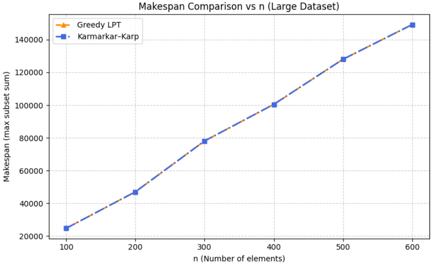


Figure: *

(b) Makespan (Δ) vs n

Observations and Analysis

- **Greedy LPT:**
  - Extremely fast; time grows slowly with input size.
  - Produces nearly balanced partitions but slightly higher cost.
- **Karmarkar–Karp:**
  - Slightly higher runtime but having less cost.
  - Performs better for large n, indicating higher heuristic precision.
- Suitable for large-scale scheduling, load balancing, and resource allocation.

References I

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

Karp, R. M. (1972). *Reducibility Among Combinatorial Problems.* In R. E. Miller and J. W. Thatcher (Eds.), *Complexity of Computer Computations*, Springer.

Karmarkar, N., and Karp, R. M. (1982). *The Differencing Method of Set Partitioning.* UCB Technical Report 82/113, University of California, Berkeley.

Mertens, S. (2001). *The Easiest Hard Problem: Number Partitioning.* Available on ResearchGate: `https://www.researchgate.net/publication/1939227_The_Easiest_Hard_Problem_Number_Partitioning`

References II

Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C. C. (1989). *Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. Artificial Intelligence, 37(1–3), 271–350.* Available via ScienceDirect: https://pdf.sciencedirectassets.com/.../main.pdf

Pedroso, J. P., and Kubo, M. (2008). *Heuristics and Exact Methods for Number Partitioning.* Technical Report, University of Porto. https://www.dcc.fc.up.pt/~jpp/publications/PDF/numpartition-DCC.pdf

A Direct Proof of the 4/3 Bound of LPT Scheduling Rule https://www.researchgate.net/publication/317108305_A_Direct_Proof_of_the_43_Bound_of_LPT_Scheduling_Rule