Partition problem

From Wikipedia, the free encyclopedia

In number theory and computer science, the **partition problem** (or **number partitioning**[1]) is the task of deciding whether a given multiset S of positive integers can be partitioned into two subsets S_1 and S_2 such that the sum of the numbers in S_1 equals the sum of the numbers in S_2. Although the partition problem is NP-complete, there is a pseudo-polynomial time dynamic programming solution, and there are heuristics that solve the problem in many instances, either optimally or approximately. For this reason, it has been called "the easiest NP-hard problem".[2]

There is an optimization version of the partition problem, which is to partition the multiset S into two subsets S_1, S_2 such that the difference between the sum of elements in S_1 and the sum of elements in S_2 is minimized. The optimization version is NP-hard, but can be solved efficiently in practice.[3]

Contents

Examples

Given $S = \{3,1,1,2,2,1\}$, a valid solution to the partition problem is the two sets $S_1 = \{1,1,1,2\}$ and $S_2 = \{2,3\}$. Both sets sum to 5, and they partition S. Note that this solution is not unique. $S_1 = \{3,1,1\}$ and $S_2 = \{2,2,1\}$ is another solution.

Not every multiset of positive integers has a partition into two halves with equal sum. An example of such a set is $S = \{2,5\}$.

Pseudo-polynomial time algorithm

The problem can be solved using dynamic programming when the size of the set and the size of the sum of the integers in the set are not too big to render the storage requirements infeasible.

Suppose the input to the algorithm is a list of the form:

$S = x_1, \ldots, x_n$

Let N be the number of elements in S. Let K be the sum of all elements in S. That is: $K = x_1 + \ldots + x_n$. We will build an algorithm that determines whether there is a subset of S that sums to $\lfloor K/2 \rfloor$. If there is a subset, then:

if K is even, the rest of S also sums to $\lfloor K/2 \rfloor$
if K is odd, then the rest of S sums to $\lceil K/2 \rceil$. This is as good a solution as possible.

Recurrence relation

We wish to determine if there is a subset of S that sums to $\lfloor K/2 \rfloor$. Let:

$p(i, j)$ be *True* if a subset of $\{ x_1, \ldots, x_j \}$ sums to i and *False* otherwise.

Then $p(\lfloor K/2 \rfloor, n)$ is *True* if and only if there is a subset of S that sums to $\lfloor K/2 \rfloor$. The goal of our algorithm will be to compute $p(\lfloor K/2 \rfloor, n)$. In aid of this, we have the following recurrence relation:

$p(i, j)$ is True if either $p(i, j-1)$ is True or if $p(i - x_j, j-1)$ is True
$p(i, j)$ is False otherwise

The reasoning for this is as follows: there is some subset of S that sums to i using numbers

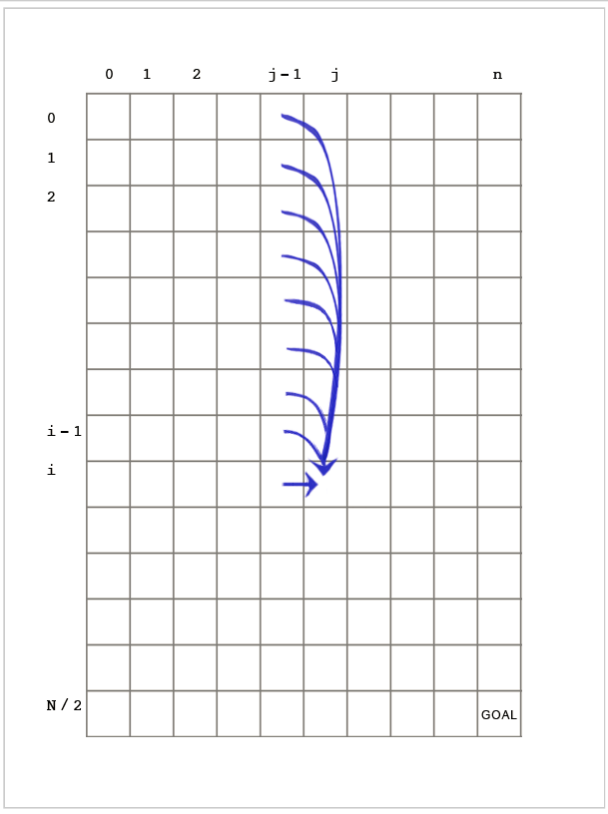
$x_1, \ldots, x_j$

if and only if either of the following is true:

There is a subset of $\{ x_1, \ldots, x_{j-1} \}$ that sums to i;
there is a subset of $\{ x_1, \ldots, x_{j-1} \}$ that sums to $i - x_j$ since x_j + that subset's sum = i.

The pseudo-polynomial algorithm

The algorithm is to build up a table of size $\lfloor K/2 \rfloor$ by n containing the values of the recurrence. Remember K is the size of the sum, while N is the number of elements. Once the entire table is filled in, return $P(\lfloor K/2 \rfloor, n)$. Below is a picture of the table P. There is a blue arrow from one block to another if the value of the target-block might depend on the value of the source-block. This dependence is a property of the recurrence relation.



Dependencies of table entry (*i*, *j*)

```
INPUT:  A list of integers S
OUTPUT: True if S can be partitioned into two subsets that have equal sum
1 function find_partition(S):
2     n ← |S|
3     K ← sum(S)
4     P ← empty boolean table of size (⌊K/2⌋ + 1) by (n + 1)
5     initialize top row (P(0,x)) of P to True
6     initialize leftmost column (P(x, 0)) of P, except for P(0, 0) to False
7     for i from 1 to ⌊K/2⌋
8         for j from 1 to n
9             if (i-S[j-1]) >= 0
10                P(i, j) ← P(i, j-1) or P(i-S[j-1], j-1)
11            else
12                P(i, j) ← P(i, j-1)
13    return P(⌊K/2⌋, n)
```

Example

Below is the table *P* for the example set used above *S* = {3, 1, 1, 2, 2, 1}:

The entry in row c and column {s1,s2,...,sk} indicates whether there is a subset of {s1,s2,...,sk} that sums to c

	{}	{3}	{3,1}	{3,1,1}	{3,1,1,2}	{3,1,1,2,2}	{3,1,1,2,2,1}
0	True	True	True	True	True	True	True
1	False	False	True	True	True	True	True
2	False	False	False	True	True	True	True
3	False	True	True	True	True	True	True
4	False	False	True	True	True	True	True
5	False	False	False	True	True	True	True

Dynamic Programming table for S={3,1,1,2,2,1}

Result of example execution of algorithm on the table P

Analysis

This algorithm runs in time $O(K\,N)$, where N is the number of elements in the input set and K is the sum of elements in the input set.

The algorithm can be extended to the k-way multi-partitioning problem, but then takes $O(n(k-1)m^{k-1})$ memory where m is the largest number in the input, making it impractical even for $k = 3$ unless the inputs are very small numbers.[3]

Special case of the subset-sum problem

The partition problem can be viewed as a special case of the subset sum problem and the pseudo-polynomial time dynamic programming solution given above generalizes to a solution for the subset sum problem.

Approximation algorithm approaches

Several heuristic algorithms exist to produce approximations to the partition optimization problem. These can be extended to linear-space exact algorithms.[3]

The greedy algorithm

One approach to the problem, imitating the way children choose teams for a game, is the *greedy algorithm*, which iterates through the numbers in descending order, assigning each of them to whichever subset has the smaller sum. This approach has a running time of $O(n \log n)$. This heuristic works well in practice when the numbers in the set are of about the same size as its cardinality or less, but it is not guaranteed to produce the best possible partition. For example, given the set S = {4, 5, 6, 7, 8} as input, this greedy algorithm would partition S into subsets {4, 5, 8} and {6, 7}; however, S has an exactly balanced partition into subsets {7, 8} and {4, 5, 6}.

This greedy approach is known to give a $^{7}/_{6}$-approximation to the optimal solution of the optimization version; that is, if the greedy algorithm outputs two sets A and B, then $\max(\sum A, \sum B) \leq \frac{7}{6}\ \mathrm{OPT}$, where OPT is the size of the larger set in the best possible partition.[4] Below is an example (written in Python) for the greedy algorithm.

```
def find_partition(int_list):
    "returns: An attempt at a partition of `int_list` into two sets of equal sum"
    A = set()
    B = set()
    for n in sorted(int_list, reverse=True):
        if sum(A) < sum(B):
            A.add(n)
        else:
            B.add(n)
    return (A, B)
```

This algorithm can be extended to the case of $k > 2$ sets: to take the k largest elements, and for each partition of them, extends the partition by adding the remaining elements successively to whichever set is smaller. (The simple version above corresponds to $k = 2$.) This version runs in time $O(2^k\, n^2)$ and is known to give a $\frac{(k+2)}{(k+1)}$ approximation.[4] Thus, we have a polynomial-time approximation scheme (PTAS) for the number partition problem, though this

is not a fully polynomial time approximation scheme (the running time is exponential in the desired approximation guarantee). However, there are variations of this idea that *are* fully polynomial-time approximation schemes for the subset-sum problem, and hence for the partition problem as well.[5][6]

Differencing algorithm

Another heuristic is the *largest differencing method* (LDM),[7] also called the *Karmarkar – Karp heuristic*[3] after the pair of scientists that published it in 1982.[8] LDM operates in two phases. The first phase of the algorithm takes the two largest numbers from the input and replaces them by their difference; this is repeated until only one number remains. The replacement represents the decision to put the two numbers in different sets, without immediately deciding which one is in which set. At the end of phase one, the single remaining number is the difference of the two subset sums. The second phase reconstructs the actual solution.[2]

The differencing heuristic performs better than the greedy one, but is still bad for instances where the numbers are exponential in the size of the set.[2]

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();
}
```

Other approaches

There are also anytime algorithms, based on the differencing heuristic, that first find the solution returned by the differencing heuristic, then find progressively better solutions as time allows (possibly requiring exponential time to reach optimality, for the worst instances).[9]

Hard instances

Sets with only one, or no partitions tend to be hardest (or most expensive) to solve compared to their input sizes. When the values are small compared to the size of the set, perfect partitions are more likely. The problem is known to undergo a "phase transition"; being likely for some sets and unlikely for others. If m is the number of bits needed to express any number in the set and n is the size of the set then $m/n < 1$ tends to have many solutions and $m/n > 1$ tends to have few or no solutions. As n and m get larger, the probability of a perfect partition goes to 1 or 0 respectively. This was originally argued based on empirical evidence by Gent and Walsh,[10] then using methods from statistical physics by Mertens,[11] and later proved by Borgs, Chayes, and Pittel.[12]

Variants and generalizations

There is a problem called the 3-partition problem which is to partition the set S into $|S|/3$ triples each with the same sum. This problem is quite different than the partition problem and has no pseudo-polynomial time algorithm unless P = NP.[13]

The *multi-way partition problem* generalizes the optimization version of the partition problem. Here, the goal is to divide a set or multiset of n integers into a given number k of subsets, minimizing the difference between the smallest and the largest subset sums.[3]

For further generalizations of the partition problem, see the Bin packing problem.

Probabilistic version

A related problem, somewhat similar to the Birthday paradox, is that of determining the size of the input set so that we have a probability of one half that there is a solution, under the assumption that each element in the set is randomly selected with uniform distribution between 1 and some given value.

The solution to this problem can be counter-intuitive, like the birthday paradox. For example, with elements randomly selected in between 1 and one million, many people's intuition is that the answer is in the thousands, tens, or even hundreds of thousands, whereas the correct answer is approximately 23 (see Birthday problem § Partition problem for details).

Notes

1. Korf 1998
2. Hayes 2002
3. Korf, Richard E. (2009). *Multi-Way Number Partitioning* (http://ijcai.org/papers09/Papers/IJCAI09-096.pdf) (PDF). IJCAI.
4. Ron L. Graham (1969). "Bounds on multiprocessor timing anomalies". *SIAM J. Appl. Math.* 17 (2). pp. 416 – 429.
5. Hans Kellerer; Ulrich Pferschy; David Pisinger (2004), *Knapsack problems* (https://books.google.com/books?id=u5DB7gck08YC&pg=PA97), Springer, p. 97, ISBN 9783540402862
6. Martello, Silvano; Toth, Paolo (1990). "4 Subset-sum problem". *Knapsack problems: Algorithms and computer interpretations*. Wiley-Interscience. pp. 105 – 136. ISBN 0-471-92420-2. MR 1086874 (https://www.ams.org/mathscinet-getitem?mr=1086874).
7. Michiels, Wil; Korst, Jan; Aarts, Emile (2003). "Performance ratios for the Karmarkar – Karp differencing method". *Electronic Notes in Discrete Mathematics*. 13: 71 – 75.
8. Karmarkar & Karp 1982
9. Korf 1998, Mertens 1999
10. Gent & Walsh 1996
11. Mertens 1998, Mertens 2001
12. Borgs, Chayes & Pittel 2001
13. Garey, Michael; Johnson, David (1979). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. pp. 96 – 105. ISBN 0-7167-1045-5.

References

- Hayes, Brian (May – June 2002), "The Easiest NP Hard Problem" (http://www.americanscientist.org/issues/pub/2002/3/the-easiest-hard-problem), *American Scientist*

- Karmarkar, Narenda; Karp, Richard M (1982), "The Differencing Method of Set Partitioning", *Technical Report UCB/CSD 82/113*, University of California at Berkeley: Computer Science Division (EECS)
- Gent, Ian; Walsh, Toby (August 1996), Wolfgang Wahlster, ed., *Phase Transitions and Annealed Theories: Number Partitioning as a Case Study* (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.4475), John Wiley and Sons, pp. 170 – 174
- Gent, Ian; Walsh, Toby (1998), "Analysis of Heuristics for Number Partitioning" (http://dx.doi.org/10.1111/0824-7935.0006), *Computational Intelligence*, 14 (3): 430 – 451, doi:10.1111/0824-7935.00069 (https://doi.org/10.1111%2F0824-7935.00069)
- Mertens, Stephan (November 1998), "Phase Transition in the Number Partitioning Problem" (http://link.aps.org/abstract/PRL/v81/p4281), *Physical Review Letters*, 81 (20): 4281, arXiv:cond-mat/9807077 (https://arxiv.org/abs/cond-mat/9807077), Bibcode:1998PhRvL..81.4281M (http://adsabs.harvard.edu/abs/1998PhRvL..81.4281M), doi:10.1103/PhysRevLett.81.4281 (https://doi.org/10.1103%2FPhysRevLett.81.4281), retrieved 2009-10-03
- Mertens, Stephan (2001), "A physicist's approach to number partitioning", *Theoretical Computer Science*, 265 (1-2): 79 – 108, doi:10.1016/S0304-3975(01)00153-0 (https://doi.org/10.1016%2FS0304-3975%2801%2900153-0)
- Mertens, Stephan (2006), "The Easiest Hard Problem: Number Partitioning", in Allon Percus; Gabriel Istrate; Cristopher Moore, *Computational complexity and statistical physics* (https://books.google.com/books?id=4YD6AxV95zEC&pg=PA125), Oxford University Press US, p. 125, arXiv:cond-mat/0310317 (https://arxiv.org/abs/cond-mat/0310317), Bibcode:2003cond.mat.10317M (http://adsabs.harvard.edu/abs/2003cond.mat.10317M), ISBN 9780195177374
- Borgs, Christian; Chayes, Jennifer; Pittel, Boris (2001), "Phase transition and finite-size scaling for the integer partitioning problem" (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.9577), *Random Structures and Algorithms*, 19 (3-4): 247 – 288, doi:10.1002/rsa.10004 (https://doi.org/10.1002%2Frsa.10004), retrieved 2009-10-04
- Korf, Richard E. (1998), "A complete anytime algorithm for number partitioning" (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.90.993), *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), retrieved 2009-10-04
- Mertens, Stephan (1999), *A complete anytime algorithm for balanced number partitioning*, arXiv:cs/9903011 (https://arxiv.org/abs/cs/9903011), Bibcode:1999cs.....3011M (http://adsabs.harvard.edu/abs/1999cs.....3011M)

Categories: NP-complete problems | Weakly NP-complete problems

---