Introduction

Homework Hotline

From proposal:
I find many surprisingly sophisticated problems lurking in the questions asked during calls at the Homework Hotline. Several tutors worked on the following problem early this year: “Suppose you had 5 rocks of unknown weight, but were given all 10 sums of weights of distinct pairs of rocks; you do not know which rocks sum to which numbers. Determine the weights of the rocks.” Actually, 10 specific weight sums were given and the 5 rock weights were to be found. In any case, the solution is to sum the 10 numbers and divide by 2: this gives the average rock weight. Observe then that the smallest number must be the sum of the smallest two rocks, similarly for the largest, and the second smallest must be the sum of the the smallest rock and the middle rock, similarly for the second largest. Now we have 5 equations, 5 unknowns. The equations happen to be linearly independent: problem solved.

Notation

[a,b,c] is a multiset, which is an unordered list of (not necessarily unique) elements.
U(A) where A is a multiset of size n returns a multiset of size nC2 which is the multiset of pairwise sums of elements of A.
A solution to S is a multiset A such that U(A) = S.
If U(A)=U(B)=S, we say that S has multiple solutions. In this case we say that both A and B lead to multiple solutions.
Unless otherwise noted, xi refers to an element of a solution, and for i<j, xi<=xj. Similarly si refers to a sum xa+xb, and for i<j, si<=sj.

Generalizations and Questions

When is there no solution? (not studied)
When are there multiple solutions? (section 2)
Upper bound on solution-finding time (section 3)
Problem complexity (not known)
Missing one weight sum (section 4)
Missing several weight sums (section 4)
Number and type of solutions desired (section 4)
Approximate solutions (section 5)

Multiple solutions

Brute force

Method: Take one solution A. See if multiple solutions to U(A) exist. Try all possible (describe!) A of size 4<=n<=13 with max xi = n+3. Try many random A with max xi = 100. Try many random A of size 14<=n<=34 with max xi = 100.

Only solutions of size 2^k lead to multiple solutions.

Construction

We can generate many solutions of size 2^k leading to multiple solutions by noting the following property: If U(A)=U(B) then for any integer m, let A'=[xi+m,xi in A] + B and let B'=[xi+m,xi in B] + A. Then U(A')=U(B') and A' and B' are twice as large as A and B. Proof: sp in U(A') = Ai+m+Aj+m = (Ai+Aj)+2m = (Br+Bs)+2m = Br+m+Bs+m = sq in U(B'). We think this produces all possible solutions leading to multiple solutions but don't know.

Basic algorithm

Bootstrap

We have that x1+x2=s1 and x1+x3=s2. It would be nice if x2+x3=s3; we could then solve for x1, x2, and x3. Unfortunately, it is possible that x1+x4<x2+x3, or indeed that x1+xi<x2+x3 for i≥4: a total of n-2 choices of weight sums resulting from x2+x3.
Suppose we have made the correct choice and know x1, x2, and x3. Remove U([x1,x2,x3]) from [si]; the smallest remaining weight is necessarily x1+x4. Since we know x1, we know x4. Similarly, if U([xi:1≤i≤k]) is removed from [si], the lowest remaining weight is necessarily x1+xk+1, and we have determined xk+1.

Algorithm 1

For each of n-2 possibilities fox x2+x3,
    Solve for x1, x2, and x3
    S = S - U([x1,x2,x3])
    For i from 4 to n,
        Let xi = min(S) – x1
        S = S – [x1+xi,x2+xi,…xi-1+xi] if possible
        If not possible, the choice was invalid
If a solution exists, at least one of the n-2 choices will have found a solution

Missing weights

One missing is symmetric

What if we are given nC2-1 weights? In many cases our algorithm would work with the minor modification that exactly once we may remove a weight from W which did not already exist in S. If the missing weight was any of r1+ri or r2+r3, however, barring duplicate weights the algorithm will fail.
Remember that we also know for N=nC2 that rn-1+rn=xN and that rn-2+rn=xN-1. A symmetric algorithm could be run that would succeed whenever the missing weight was not any of ri+rn or rn-2+rn-1. Only if the missing weight was r1+rn could both algorithms fail to find the solution, but we can include a special case

to take care of the problem: if step 5. fails on i=n, backtrack to step 4. and change it to ri=min(S)-r2.

Algorithm 2

```
For each of n-2 possibilities fox x2+x3,
    Solve for x1, x2, and x3
    S = S - U([x1,x2,x3])
    Let numSkipped = 0
    For i from 4 to n,
        Let xi = min(S) – x1
        If there is a single value v in [x1+xi,x2+xi,…xi-1+xi] not in S,
            If numSkipped<1
                add v to S
                increment numSkipped
            otherwise continue
        S = S – [x1+xi,x2+xi,…xi-1+xi] if possible
        If not possible,
            Let xi = min(S) – x2
            If there is a single value v in [x2+xi,x3+xi,…xi-1+xi] not in S,
                If numSkipped<1
                    add v to S
                    increment numSkipped
                otherwise continue
            S = S – [x2+xi,x3+xi,…xi-1+xi] if possible
            If not possible continue
For each of n-2 possibilities fox xn-2+xn-1,
    Solve for xn-2, xn-1, and xn
    S = S - U([xn-2,xn-1,xn])
    For i from n-3 to 1,
        Let xi = max(S) – xn
        S = S – [xi+xi+1,xi+xi+2,…xi+xn] if possible
        If not possible, the choice in 1. was invalid
```

If a solution exists, at least one of the 2(n-2)+1 choices will have found a solution

Two missing cases

Consider being given nC2-2 weights. If at most one of the missing weights is r1+ri, ri+rn, r2+r3, rn-2+rn-1, r2+rn, or r1+rn-1 the algorithm will still definitely succeed. We can modify the algorithm again to allow success even if both weights are problematic: simply guess which two weights among the O(n) problematic weights are missing. The modifications are messy but relatively straightforward, but we have incurred an additional O(n2) outer loop. The difficulty grows exponentially with additional missing weights.

Algorithm 3

If we include a case in our algorithm for every possible missing weight combination, then if a solution exists, after making all choices we will have found it. The common-case problem is when the missing weights are r1+ri, rj+rn. Suppose we run the symmetric algorithms until the missing case. We will have found xk for k<i and k>j.

More than two missing

With more than two missing we have to keep guessing… with at least n-1 missing we might have absolutely zero chance to get a certain solution. Need to try the case where we guess intervals in which the sums are missing, i.e., the missing sum would be the fifth if we had it. Then it's just (nC2)Ck which is a little better but still very exponential.

Amortized times

Bad cases occur when lots of the missing weights happen to be the wrong missing weights. The worst cases occur with very small probability, which helps certain algorithm, just not the ones that try to find all solutions. We can start, for example, by assuming the least-work missing structure, then solving from there. Give a graph of running times here for n, k, and the two types of problems. We don't know enough about the frequency or distribution of solutions to analyze the third type.

Large instances

PSO

Particle Swarm Optimization would find exact solutions in a convex space which this decidedly is not. The results thus far are slow convergence to a non-optimal solution. My conjecture is that this is a result of the components not being independent, since the distance metric allows permutations. Plus there are the huge numbers of local minima and the information we simply discard.

Heuristic

Consider two collections S and T where T has k missing weight sums. Find a function f that takes elements in T to elements in S. Define the distance between S and T to be the minimum of sum(abs(i,f(i)),i indexing T) over all such functions. DP allows this to be done in O(nk). Using this distance function, as mentioned, allows element permutation and doesn't use all of S so that changing a 3 to a 3000, say, may not change the distance.

Genetic

Genetic algorithms may work better because they view the solution space as composed of subsolutions instead of being like a traversable landscape. Our solutions really are composed of good smaller solutions in that a good collection of values, one which, say, shares many summed elements in common with the target, will help out most other possible collections.

Future work

Questions

Do only problems generated from solutions of size 2^k have multiple solutions? If so, can all such problems be constructed using the symmetric addition described?
Can we learn anything from the structure of A in Ax=Pb? (Note: where does this fit?)
When many weights are missing, can we find deterministic algorithms better than simply guessing which weights are missing? Especially this arbitrarily choosing positions rather than two solution variables.
When weights are removed, how many new solutions appear? Which weights do what? How much information is really stored by the middle weights?
Can we find a better distance function?
Can evolutionary algorithms do better than PSO? (mine)