Behind FLSSS

Charlie Wusuo Liu

Abstract

The design of the R package FLSSS, Fixed Length Subset Sum Solver with error specification, and its extension to multidimensional domain and to the general-purpose Knapsack Problem. If the animations cannot run properly in web browser, please read the article in Adobe or other PDF viewer.

Contents

I. Introduction

Well, although typing in Latex, I definitely do not want to write it like a publishable paper. Instead, I would like to go for a personal memo style. The more academic papers I read nowadays, the more often I am annoyed by those playing "sophisticated" games like hide–and–seek, putting out fancy irrelevance, using over–complicated notations, covering simple math with a big pile of garbage... Where are all the suggestive and unpretentious forms of knowledge? Sigh. I must have not read enough to gain the capability of seeing through these camouflages without having a headache.

I will try to explain everything as intuitive as possible. However, math equations, algorithm tables are inevitable because the details will be kicking in sooner or later. To resolve the conflict, I write section IV "The Big Picture". For those who do not want to dive in the skills and proofs, this section alone can provide enough information on the algorithmic framework, at least for the single–dimensional scenario.

Hope the article can interest you in the topic or inspire you for solving problems of your own.

And special thanks to Dirk Eddelbuettel, Romain Francois and other contributors of R package Rcpp [1]; JJ Allaire, Romain Francois and other contributors of R package RcppParallel [3]; JJ Allaire and Hadley Wickham, founder and chief scientist at Rstudio, Inc [4]. Their contributions made coding and compiling C++ for R package production extremely easy and smooth, and undoubtedly, pushed the power of R to a completely new level. And thanks to Henrik Bengtsson, creator of the R package R.rsp [5] which is a nice tool able to easily build R package with static PDF vignette.

II. Motivation

Combinatorics always interests me. It can be extremely challenging yet anyone with high school education is able to mess with it. Like Terrence Tao said, it is the field which still has large areas closely resembling its classical roots without being informed by modern algebra. So, average people like me, grasp the fun while you still can!

During years I often feel many hard statistical problems eventually come down to the combinatorics. You can avoid confronting it by all means in statistics, but the shadow of the mountain would continuously remind you that you have never truly climbed over it, even after you went far away enough that the mountain shrank to a pin, in the butt!

When I was still processing filings in Division of Insurance, once I got the annual losses of about 100 insurers and the aggregated loss of 4 companies to be investigated, whose names were unknown yet. While waiting for more information to come in, I tried programming to find those babies out because the loss values looked pretty unique so it should not be too hard to uncover which summed up to the given total. Well then, once the gate got open, things pouring out, hard to close ever since.

III. FLSSS

Let's review the problem:

Given a real set/vector $X = (x_1, x_2, \ldots, x_N)$, a target value T, a subset/subvector size L where $L \le N$, an error E, find one or more subsets of X of size L like $X_c = (x_{c_1}, x_{c_2}, \ldots, x_{c_L})$, such that $\sum_{i=1}^{L} x_{c_i} \in [T - E, T + E] = [\text{Min}, \text{Max}]$.

We will restrict X to be a sorted vector so $x_1 \le x_2 \ldots \le x_N$, which is important. A fancy view of my algorithm's essence is it would drain every advantage of the reduction in the information entropy caused by this sorting. Am I talking like fabulous or what!

Since X is sorted, we additionally focus only on the subset's index vector $C = (c_1, c_2 \ldots, c_L) \subset (1, 2, \ldots, N)$ and also restrict $c_1 < c_2 < \ldots < c_L$. Two important properties on index vector are:

- every subset of X is associated to a unique index vector.
- for any $c_i, c_j \in (1, 2, \ldots, N)$, we have $c_i < c_j \implies x_{c_i} \le x_{c_j}$.

Define S as the operator for summing up elements in X's subset indexed by C. Then,

- $C = (c_1, c_2, \ldots, c_L)$ is a qualified solution $\iff S(c_1, c_2, \ldots, c_L) \in [\text{Min}, \text{Max}]$.
- for another index vector $C' = (c'_1, c'_2, \ldots, c'_L)$, we have
  $c_1 < c'_1, c_2 < c'_2, \ldots, c_L < c'_L \implies$
  $S(c_1, c_2, \ldots, c_L) \le S(c'_1, c'_2, \ldots, c'_L)$

IV. The Big Picture

Remember the Sandwich Theorem from Calculus? FLSSS has the same taste. It follows a paradigm similar to branch–and–bound: we squeeze the combinatorial space, or the initial solution space, over and over, until it will not shrink anymore. Then we take a lower–dimensional subspace, do the evil again. Eventually we can or we cannot squeeze out some solution(s).

So, there are two major components in the algorithm. One is to squeeze the combinatorial space, the other is to take a lower–dimensional subspace.

The domain of index vector $C = (c_1, c_2, \ldots, c_L)$ is seen as the initial solution space. Let's look at it element–wisely: because of the restriction $c_1 < c_2 < \ldots < c_L$, the lower bounds for all elements in C should be $(1, 2, \ldots, L)$ and the upper bounds should be $(N - L + 1, N - L + 2, \ldots, N)$.

In other words, c_1's domain is the integer sequence $1 : N-L+1$, c_2's domain is the integer sequence $2 : N-L+2$, $\ldots$, and c_L's domain is the integer sequence $L : N$.

We call $(1, 2, \ldots, L)$ "lower bound vector" and $(N - L + 1, N - L + 2, \ldots, N)$ "upper bound vector".

Given the initial bounding vectors which are $(1, 2, \ldots, L)$ and $(N - L + 1, N - L + 2, \ldots, N)$, the squeezing procedure aims to pull up elements in the lower bound vector and to pull down those in the upper bound vector as much as possible, during which the restriction $c_1 < c_2 < \ldots < c_L$ must not be violated.

One may ask based on what can we squeeze the bounds? The answer is the prerequisite subset sum range [Min, Max]. To have the subset sum indexed by $c_1, c_2, \ldots, c_L$ bounded by [Min, Max], simple logic tells us none of the elements $c_i, i \in (1, 2, \ldots, L)$ can be unbounded, and the bound is not necessarily the initial one: $[i, N - L + i]$. For more details, please read section V.

The squeezing procedure runs in an iterative fashion. Soon, it will sense the lower and upper bounds are no longer updated. At this time we call the bounding vectors infimum (greatest lower bound) and supremum (lowest upper bound) vectors, denoted by $C^{\inf} = (c_1^{\inf}, c_2^{\inf}, \ldots, c_L^{\inf})$ and $C^{\sup} = (c_1^{\sup}, c_2^{\sup}, \ldots, c_L^{\sup})$.

If $C^{\inf} = C^{\sup}$, $C^{\inf}$ is a solution.

When the bounding vectors are unequal yet cannot be updated anymore, we initialize a depth/best–first search, which prompts the subspacing procedure. In short, we select an $i \in (1, 2, \ldots, L)$ and loop c_i over its domain: integer sequence $c_i^{\inf} : c_i^{\sup}$. Within each iteration, c_i is fixed and we focus on the $L - 1$ dimensional subspace $(c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_L)$. Another set of initial bounding vectors of size $L - 1$ can be obtained through appropriate calculation.

Naturally, we again impose squeezing, approaching the $L-2$ dimensional subspace to continue the vicious cycle. In any subspace, we bounce back to the parent if no solution or not enough solutions are found from itself and the children.

Choosing the best $i \in (1, 2, \ldots, L)$ for subspacing is a typical heuristic. Intuitively, selecting i such that c_i has the least width of domain, or $c_i^{\sup} - c_i^{\inf}$, will expand the least number of searching branches, or subspaces.

Under this criterion, the best situation for subspacing would be having an i where $c_i^{\inf} = c_i^{\sup}$, meaning a squeezing call can simply reduce the combinatorial space by 1 dimension. However this does not mean the succeeding branchings and the current one will line up as the global optimal sequence of subspacing. What is more, the best sequence of subspacing for dimension reduction does not necessarily mean it will find the **first** solution in the shortest time.

Discussion above can gradually become clearer in the way to section VI. The current version of FLSSS and its extension uses the least width of domain as the criterion to pick i.

V. Squeezing

As mentioned, we update the bounding vectors iteratively. More specifically, we update the lower bound vector according to the upper bound vector and vice versa, like the E–step and M–step in the Expectation Maximization algorithm.

Keep in mind we want the subset elements to sum in $[\text{Min}, \text{Max}]$, and we have defined function

$$S(c_1, c_2, \ldots, c_L) = \sum_{k=1}^{L} x_{c_k} \tag{1}$$

When the upper bound vector $C^{\text{UB}} = (c_1^{\text{UB}}, c_2^{\text{UB}}, \ldots, c_L^{\text{UB}})$ is known, the lowest possible value c_i can hit is when the rest elements, $(c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_L)$, reach their individual greatest possible values, which is easy to understand by looking at the following inequality:

$$\begin{aligned} &S(c_1, \ldots, c_i, \ldots, c_L) \geq \text{Min} \iff \\ &S(c_i) \geq \text{Min} - S(c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_L) \end{aligned} \tag{2}$$

One might think $(c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_L)$'s individual greatest values should just be their current upper bounds listed in C^{UB}. However, this thinking ignores the primary restriction $c_1 < c_2 < \ldots < c_L$, which indicates c_i itself caps $c_1, c_2, \ldots, c_{i-1}$. Therefore, the correct greatest value for $c_{j\{j<i\}}$ should be $\min\big(c_i - (i - j),\, c_j^{\text{UB}}\big)$, and for $c_{j\{j>i\}}$, it is simply c_j^{UB}.

Based on eq. 2, the lower bound of c_i, or c_i^{LB}, must meet the following two requirements:

$$\begin{aligned} &S\Big(\min\big(c_i^{\text{LB}} - (i-1),\, c_1^{\text{UB}}\big), \ldots, \min\big(c_i^{\text{LB}} - 1,\, c_{i-1}^{\text{UB}}\big)\Big) \\ &+ S(c_i^{\text{LB}}) \geq \text{Min} - S\big(c_{i+1}^{\text{UB}}, \ldots, c_L^{\text{UB}}\big) \\ \\ &S\Big(\min\big(c_i^{\text{LB}} - 1 - (i-1),\, c_1^{\text{UB}}\big), \ldots, \min\big(c_i^{\text{LB}} - 1 - 1,\, c_{i-1}^{\text{UB}}\big)\Big) \\ &+ S(c_i^{\text{LB}} - 1) < \text{Min} - S\big(c_{i+1}^{\text{UB}}, \ldots, c_L^{\text{UB}}\big) \end{aligned} \tag{3}$$

The two inequalities in 3 guarantees that, from low to high, c_i^{LB} is the first value in c_i's domain that can make the subset sum $\geq$ Min while $c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_L$ are at their individual greatest possible values.

I doubt there exists an analytical solution to system 3, but if my work is ever gonna attract some math genius proving me wrong, the world would be much more fun!

A. *Locating c_i^{LB} and Engineering*

Based on the explanation, you might already have the idea of linear searching for c_i^{LB}: start from the current lower bound of c_i, increment it by 1 each time until the first inequality in 3 becomes true.

Some early versions of FLSSS used linear search with some improvements, making it efficient enough to be released. Speaking of which, I would claim the implementation of this algorithm is so vital to the speed that it almost belittles the mathematics. A good implementation must well consider the details in memory management, object efficiency and complier friendliness. A better data structure or a smarter way of programming could lead to speed increase in the order of magnitude due to the algorithmic complexity. Through the article not all the important engineering details will be touched, and better ones may be discovered later then the package will be updated accordingly.

We will first discuss linear search. It helps clear the concepts and proofs, based on which a binary search method with an auxiliary data structure will then be introduced.

B. *Linear Method*

Starting from the current lower bound of c_i, each time we increment c_i as $c_i \leftarrow c_i + 1$, we exam inequality 4:

$$S\Big(\min\big(c_i - (i-1),\, c_1^{\text{UB}}\big), \ldots, \min\big(c_i - 1,\, c_{i-1}^{\text{UB}}\big)\Big) \\ + S(c_i) \geq \text{Min} - S\big(c_{i+1}^{\text{UB}}, \ldots, c_L^{\text{UB}}\big) \tag{4}$$

Once it becomes true, c_i becomes the new lower bound.

Valuing ineq. 4 directly would be unnecessarily heavy. Some work can be done to reduce the complexity, and 4 will eventually become 8.

Observing ineq. 4, the right side does not change while incrementing c_i. On the left side, an important property is

$$\begin{aligned} &\text{If } j \in [1,\, i] \text{ and} \\ &\qquad \min\big(c_i - (i-j),\, c_j^{\text{UB}}\big) = c_j^{\text{UB}}, \\ &\text{then for any } k \leq j, \\ &\qquad \min\big(c_i - (i-k),\, c_k^{\text{UB}}\big) = c_k^{\text{UB}} \text{ is true} \end{aligned} \tag{5}$$

The proof of property 5 is showed in the following box. Feel free to skip it if not interested.

Consider the primary restriction $c_1 < c_2 < \ldots < c_i$ and the fact that they are integers, we have $c_j - c_k \geq j - k$, which leads to $c_j^{\text{UB}} - c_k^{\text{UB}} \geq j - k$. If $c_i - (i-j) \geq c_j^{\text{UB}}$ is true, then $c_i - (i-j) + (k-j) \geq c_j^{\text{UB}} + (c_k^{\text{UB}} - c_j^{\text{UB}})$ is also true. Therefore, $\min\big(c_i - (i-k),\, c_k^{\text{UB}}\big) = c_k^{\text{UB}}$

In plain words, property 5 states that during incrementing c_i, if any one on c_i's left has reached its individual upper bound, then everyone on that one's left have also reached their individual upper bounds.

A picture is said worth a thousand words. I guess an animation is worth ten thousand. Figure 1 may provide an intuition of the practical meaning of property 5 and its contrapositive, which is showed in below:

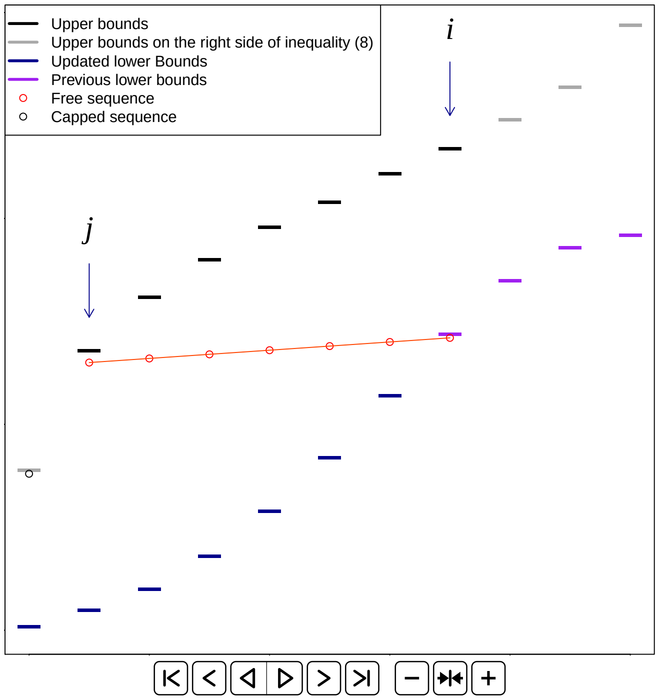

Fig. 1: Linear Method for c_i^{LB}

$$\begin{aligned} &\text{If } j \in [1,\, i] \text{ and} \\ &\qquad \min\big(c_i - (i-j),\, c_j^{\text{UB}}\big) = c_i - (i-j), \\ &\text{then for any } k \text{ where } j \leq k \leq i, \\ &\qquad \min\big(c_i - (i-k),\, c_k^{\text{UB}}\big) = c_i - (i-k) \text{ is true} \end{aligned} \tag{6}$$

In plain words, property 6 says if any one on the left of c_i has not reached its individual upper bound, then everyone between that one and c_i have not reached their individual upper bounds either.

Properties 5 and 6 reveal a fact: during incrementing c_i, there always exists a $j \in [1,\, i]$ where $c_1, \ldots, c_{j-1}$ have reached their upper bounds and $c_j, \ldots, c_i$ have not. In other words, the left side of ineq. 4 can be written as

$$\begin{aligned} &S\Big(\min\big(c_i - (i-1),\, c_1^{\text{UB}}\big), \ldots, \min\big(c_i - 1,\, c_{i-1}^{\text{UB}}\big),\, c_i\Big) \\ =&S\Big(c_1^{\text{UB}}, \ldots, c_{j-1}^{\text{UB}},\, c_i - (i-j), \ldots, c_{i-1},\, c_i\Big) \\ =&S\Big(c_1^{\text{UB}}, \ldots, c_{j-1}^{\text{UB}}\Big) + S\Big(c_i - (i-j), \ldots, c_i\Big) \end{aligned} \tag{7}$$

Thus, ineq. 4 becomes

$$\overbrace{S\big(c_i - (i-j), \ldots, c_i\big)}^{\mathbb{S}_{\text{L}}} \geq \\ \underbrace{\text{Min} - S\big(c_{i+1}^{\text{UB}}, \ldots, c_L^{\text{UB}}\big) - S\big(c_1^{\text{UB}}, \ldots, c_{j-1}^{\text{UB}}\big)}_{\mathbb{S}_{\text{R}}} \tag{8}$$

We will refer j as the "freedom point" in the rest of the article.

Valuing ineq. 8 has a good advantage. Let $\mathbb{S}_\mathrm{L}$ be the left side sum in ineq. 8. Instead of recalculating $\mathbb{S}_\mathrm{L} \leftarrow S\big(c_i - (i-j), \dots, c_i\big)$ after $c_i \leftarrow c_i + 1$, one can update $\mathbb{S}_\mathrm{L}$ by $\mathbb{S}_\mathrm{L} \leftarrow \mathbb{S}_\mathrm{L} - S\big(c_i - (i-j)\big) + S\big(c_i + 1\big)$ before c_i is incremented, because the consecutive sequence $c_i - (i-j) + 1 : c_i$ before c_i incremented will be exactly the same as the sequence $c_i - (i-j) : c_i - 1$ after c_i incremented.

In other words, prior to c_i being incremented, the new sum $\mathbb{S}_\mathrm{L}$ can be computed by simply subtracting the tail $S\big(c_i - (i-j)\big)$ then adding the new head $S\big(c_i+1\big)$. One can see the amount of arithmetic is significantly reduced compared to naively repeating the sequence summation for updating $\mathbb{S}_\mathrm{L}$.

Denote the right side value of ineq. 8 by $\mathbb{S}_\mathrm{R}$. The freedom point j could change throughout finding c_i^LB and $\mathbb{S}_\mathrm{R}$ need be changed correspondingly. When c_i is increased to the extent that $c_i - (i-j) = c_j^\mathrm{UB}$, meaning c_j reaches its upper bound, and if ineq. 8 is still false, we update $\mathbb{S}_\mathrm{R}$ by $\mathbb{S}_\mathrm{R} \leftarrow \mathbb{S}_\mathrm{R} - S(c_j^\mathrm{UB})$ then increment $j \leftarrow j + 1$.

Once c_i^LB is found, increment $i \leftarrow i + 1$ to continue. Algorithm 1 and 2 summarize the linear search method for locating c_1^LB and $c_{i,\,i\geq 2}^\mathrm{LB}$ respectively.

Algorithm 1 Linear search for c_1^LB

```
KNOWN:
    Previous lower bound $c_1^{\mathrm{LBp}}$
    All the upper bounds $(c_1^\mathrm{UB}, \dots, c_L^\mathrm{UB})$
    Sum of the upper bounds $\mathbb{S}^\mathrm{UB} = S(c_1^\mathrm{UB}, \dots, c_L^\mathrm{UB})$
COMPUTE: New lower bound $c_1^\mathrm{LB}$.
1: Initialize $c_1 \leftarrow c_1^{\mathrm{LBp}}$, $\mathbb{S}_\mathrm{L} \leftarrow S(c_1)$, $\mathbb{S}_\mathrm{R} \leftarrow \mathrm{Min} - \mathbb{S}^\mathrm{UB} + S(c_1^\mathrm{UB})$
2: loop
3:   if $\mathbb{S}_\mathrm{L} < \mathbb{S}_\mathrm{R}$ then
4:     if $c_1 = c_1^\mathrm{UB}$ then
5:       return. ◁ solution does not exist
6:     end if
7:     $c_1 \leftarrow c_1 + 1$
8:     $\mathbb{S}_\mathrm{L} \leftarrow S(c_1)$
9:   else
10:    $c_1^\mathrm{LB} \leftarrow c_1$; break
11:  end if
12: end loop
```

C. *Locate c_i^{UB}*

In the same rationale of finding c_i^LB, c_i^UB should be the greatest integer when $(c_1, \dots, c_{i-1}, c_{i+1}, \dots, c_L)$ are at their individual lower bounds while the subset sum is no greater than Max. The definitional inequalities 3 for c_i^LB

Algorithm 2 Linear search for c_i^LB, $i \geq 2$

```
KNOWN: $c_{i-1}^\mathrm{LB}$, $c_i^\mathrm{LBp}$, $\mathbb{S}_\mathrm{L}$, $\mathbb{S}_\mathrm{R}$, $j$, $(c_1^\mathrm{UB}, \dots, c_L^\mathrm{UB})$
COMPUTE: New lower bound $c_i^\mathrm{LB}$.
1: Initialize $c_i \leftarrow c_i^\mathrm{LBp}$
2: if $c_i \leq c_{i-1}^\mathrm{LB} + 1$ then
3:   $c_i \leftarrow c_{i-1}^\mathrm{LB} + 1$
4:   $\mathbb{S}_\mathrm{R} \leftarrow \mathbb{S}_\mathrm{R} + c_i^\mathrm{UB}$
5:   $\mathbb{S}_\mathrm{L} \leftarrow \mathbb{S}_\mathrm{L} + c_i$
6: else
7:   $k \leftarrow c_i - (c_{i-1}^\mathrm{LB} + 1)$
8:   while $c_i - (i-j) + k \geq c_j^\mathrm{UB}$ do
9:     $\mathbb{S}_\mathrm{R} \leftarrow \mathbb{S}_\mathrm{R} - S\big(c_j^\mathrm{UB}\big)$
10:    $j \leftarrow j + 1$
11:  end while
12:  $\mathbb{S}_\mathrm{L} \leftarrow S\big(c_i - (i-j), \dots, c_i\big)$
13: end if
14: loop
15:   if $\mathbb{S}_\mathrm{L} < \mathbb{S}_\mathrm{R}$ then
16:     if $c_i = c_i^\mathrm{UB}$ then
17:       return. ◁ solution does not exist
18:     end if
19:     $\mathbb{S}_\mathrm{L} \leftarrow \mathbb{S}_\mathrm{L} - S\big(c_i - (i-j)\big) + S(c_i + 1)$
20:     $c_i \leftarrow c_i + 1$
21:     while $c_i - (i-j) = c_j^\mathrm{UB}$ do
22:       $\mathbb{S}_\mathrm{L} \leftarrow \mathbb{S}_\mathrm{L} - S\big(c_i - (i-j)\big)$
23:       $\mathbb{S}_\mathrm{R} \leftarrow \mathbb{S}_\mathrm{R} - S(c_j^\mathrm{UB})$
24:       $j \leftarrow j + 1$
25:     end while
26:   else
27:     $c_i^\mathrm{LB} \leftarrow c_i$
28:     $i \leftarrow i + 1$
29:   end if
30: end loop
```

can be adjusted to inequalities 9 for defining c_i^UB.

$$\begin{aligned}
&S\Big(\max\big(c_i^\mathrm{UB} + 1,\, c_{i+1}^\mathrm{LB}\big), \dots, \max\big(c_i^\mathrm{UB} + L - i,\, c_L^\mathrm{LB}\big)\Big) \\
&\quad + S(c_i^\mathrm{UB}) \leq \mathrm{Max} - S\big(c_1^\mathrm{LB}, \dots, c_{i-1}^\mathrm{LB}\big) \\[2ex]
&S\Big(\max\big(c_i^\mathrm{UB} + 2,\, c_{i+1}^\mathrm{UB}\big), \dots, \max\big(c_i^\mathrm{UB} + 1 + L - i,\, c_L^\mathrm{UB}\big)\Big) \\
&\quad + S(c_i^\mathrm{UB} + 1) > \mathrm{Max} - S\big(c_1^\mathrm{LB}, \dots, c_{i-1}^\mathrm{LB}\big)
\end{aligned} \tag{9}$$

We go in the reverse direction for locating the upper bounds, in other words, compute c_i^UB after c_{i+1}^UB found.

D. *Cumulative Sum Matrix*

From inequality 8 or animation 1, one can see we repetitively value sum of the subset indexed by consecutive sequence $c_i - (i-j), \dots, c_i$. The size of the sequence ranges from 1 to L where L is the subset size. Considering that the superset X has N elements, the number of different consecutive sequences representing a subset of size $k \in [1, L]$ is $N - k + 1$. We write all the subset sums of size $k = 1, \dots, L$ indexed by a consecutive sequence in a quasi–triangle matrix $\mathcal{M}$, showed in fig. 2. In this matrix,

$$\begin{pmatrix} x_1 & x_1+x_2 & \sum_{t=1}^{3} x_t & \dots & \sum_{t=1}^{L} x_t \\ x_2 & x_2+x_3 & \sum_{t=2}^{4} x_t & \dots & \vdots \\ \vdots & \vdots & \vdots & & \sum\limits_{t=N-L+1}^{N} x_t \\ x_{N-2} & x_{N-2}+x_{N-1} & \sum\limits_{t=N-2}^{N} x_t & & \\ x_{N-1} & x_{N-1}+x_N & & & \\ x_N & & & & \end{pmatrix}$$

Fig. 2: Cumulative Sum Matrix $\mathcal{M}$

column k contains the sums of all possible subsets indexed by a consecutive sequence of size k in ascending order.

Matrix $\mathcal{M}$ will be used to support a binary search method for c_i^{LB} and c_i^{UB}.

E. *Binary Method for c_i^{LB}*

Using $\mathcal{M}$, we will first find the freedom point j, that is, we can predict the value of j when inequality 8 becomes true. But beforehand we will discuss what happens next once j is found .

Entries in each column of $\mathcal{M}$ are naturally ordered from low to high because $x_1 \le x_2 \le \ldots \le x_N$. The consecutive sequence sum $\mathbb{S}_{\mathrm{L}} = S\big(c_i - (i-j), \ldots, c_i\big)$ must be lying in column $i-j+1$. Therefore, we can apply a binary search on column $i-j+1$ to locate index of the lowest $\mathbb{S}_{\mathrm{L}}$ that satisfies ineq. 8. Assume the index is found to be k, c_i^{LB} will immediately be known as $c_i^{\mathrm{LB}} \leftarrow i - j + k$.

F. *Locate Freedom Point*

Remember in the linear search method, j can only go in one direction: being incremented. This tells us we can directly increment the previous j by one at a time to detect the new j. Animation 3 illustrates the logic. In fact, the new j must be the first j from low to high that makes ineq. 8 come true.

Like the animation shows, each time j is incremented, the free consecutive sequence sum $\mathbb{S}_{\mathrm{L}}$ should be recalculated and it must be equal to $\mathcal{M}[c_{j-1}^{\mathrm{UB}} + 1,\ i - j + 1]$. Meanwhile, $S(c_{j-1}^{\mathrm{UB}})$ should be subtracted from $\mathbb{S}_{\mathrm{R}}$. Then inequality 8 is evaluated again. More details can be found in algorithm 3.

VI. Subspacing

As discussed in section "The Big Picture", after the greatest lower bound and lowest upper bound vectors, C^{inf} and C^{sup}, are found, in the current version of FLSSS, we select the i such that the least searching branches will be spawn, or $i = \arg\min\limits_{i\in[1,\,L]} (c_i^{\mathrm{sup}} - c_i^{\mathrm{inf}})$ for subspacing. More specifically, we fix c_i to each of the elements in integer sequence $c_i^{\mathrm{inf}} : c_i^{\mathrm{sup}}$, and repeat the squeezing in the one-less dimensional combinatorial space.

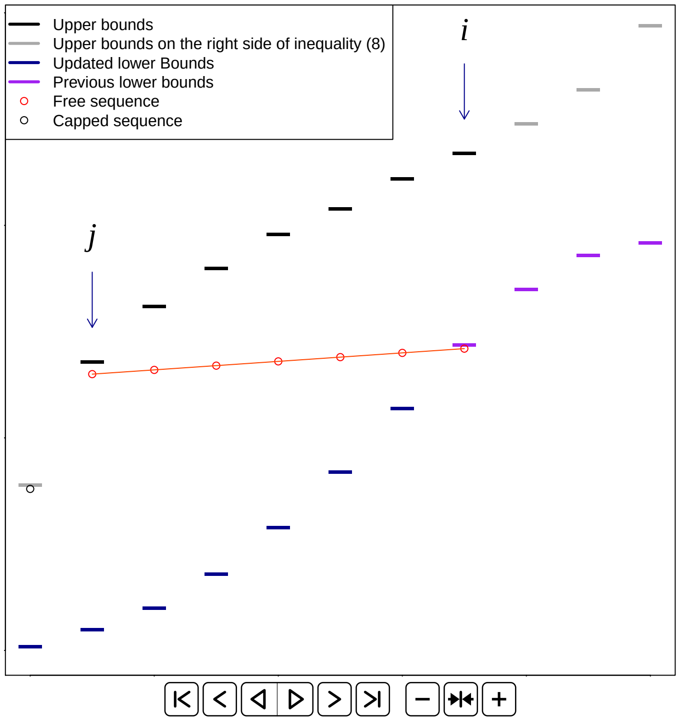

Fig. 3: Binary Method for c_i^{LB}

A. *Give Birth*

The child space (subspace) will inherit the bounding vectors from its parent with certain adjustment. When c_i is fixed to an integer in $c_i^{\mathrm{inf}} : c_i^{\mathrm{sup}}$, equivalently we are imposing $c_i^{\mathrm{LB}} = c_i^{\mathrm{UB}} = c_i$. Due to the primary restriction $c_1 < c_2 < \ldots < c_L$, the upper bounds on c_i's left, or $c_1^{\mathrm{UB}}, \ldots, c_{i-1}^{\mathrm{UB}}$, should all be less than c_i, and the lower bounds on c_i's right, or $c_{i+1}^{\mathrm{LB}}, \ldots, c_L^{\mathrm{LB}}$, should all be greater than c_i. We need to refresh the inherited bounding vectors in the child space after squeezing.

The earliest version of FLSSS does explicit recursion for the depth/best-first search. Later it was reimplemented as an iterative process via a stack scheme. The speed and space efficiency gain is significant.

In the algorithm, each subspace is an object of 9 members:

- i, as the optimal i for subspacing in the parent space.
- c_i^{sup}, the supremum for c_i computed in this space.
- c_i, the current fixation in the sequence of $c_i^{\mathrm{inf}} : c_i^{\mathrm{sup}}$.
- T, the target subset sum for this space.
- C^{LB} and C^{UB}, the bounding vectors for this space.
- $\mathbb{S}^{\mathrm{LB}}$ and $\mathbb{S}^{\mathrm{UB}}$, sums of subsets indexed by the bounding vectors.
- C^{UBLR}. After the squeezing procedure and the optimal i is selected, C^{UBLR} is a copy of elements $c_1^{\mathrm{sup}} : c_{i-1}^{\mathrm{sup}}$. "UBLR" stands for "upper bound left reserve". Further explanation will be made later.

Each space object gives birth to the child space in stack, as explained in algorithm 4.

Algorithm 3 Binary search for c_i^{LB}, $i \geq 2$

KNOWN: c_{i-1}^{LB}, c_i^{LBp}, $\mathbb{S}_{\text{R}}$, j, $(c_1^{\text{UB}}, \ldots, c_L^{\text{UB}})$, B_{srch}:=binary search function, $\mathcal{M}$ (cumulative sum matrix).
NOTE: When $i = 1$, follow algorithm 1 and set $j = 1$.
COMPUTE: New lower bound c_i^{LB}.

```
1: Initialize $c_i \leftarrow c_i^{\text{LBp}}$
2: if $c_i \leq c_{i-1}^{\text{LB}} + 1$ then
3:     $c_i \leftarrow c_{i-1}^{\text{LB}} + 1$
4: end if
5: loop
6:     if $i < j$ then
7:         return. $\triangleleft$ solution does not exist
8:     end if
9:     if $c_i - (i-j) \geq c_j^{\text{UB}}$ or $\mathcal{M}[c_i - (i-j),\, i-j+1] < \mathbb{S}_{\text{R}}$ then
10:        $\mathbb{S}_{\text{R}} \leftarrow \mathbb{S}_{\text{R}} - c_j^{\text{UB}}$
11:        $j \leftarrow j + 1$
12:    else break
13:    end if
14: end loop
15: $u \leftarrow \&\big[\mathcal{M}[c_{j-1}^{\text{UB}} + 1,\, i-j+1]\big]$ $\triangleleft$ & maps memory address
16: $v \leftarrow \&\big[\mathcal{M}[c_i - (i-j),\, i-j+1]\big]$
17: $I_\theta \leftarrow \text{B}_{\text{srch}}(u, v, \mathbb{S}_{\text{R}})$ $\triangleleft$ assume $\text{B}_{\text{srch}}()$ takes in the beginning and the end of a sorted sequence, returning the index of the least element no less than $\mathbb{S}_{\text{R}}$.
18: $c_i^{\text{LB}} \leftarrow i - j + I_\theta$
```

B. *Update*

If the child does not return enough solutions, the parent will update itself, explained in algorithm 5, and then give birth to the next child as replacement.

A number of details are omitted from algorithm 4 and 5. For example, the scenarios when $i = 1$ and $i = L$ are treated differently to gain speed and space efficiency.

If no more children can be born or enough solutions have been received, the parent will be cleared from stack and the function will return to the grandparent level.

C. *Does $\mathcal{M}$ Still Work in Subspace*

If you managed to understand everything all the way down here, something might strike you suddenly: after c_i^{UB} and c_i^{LB} are erased from C^{UB} and C^{LB}, it seems the subspace becomes nonconsecutive with respect to the parent space, so will $\mathcal{M}$, which stores the sums of consecutive sequences in the parent space, work as it is supposed to?

The answer is yes. Remember when c_i^{UB} and c_i^{LB} are erased, the new shorter bounding vectors for the subspace are also refreshed correspondingly. Squeezing them is absolutely no different than squeezing the parent's bounding vectors with $c_i^{\text{UB}} = c_i^{\text{LB}} = c_i$ enforced. Thus $\mathcal{M}$ can stay intact and is applied to the squeezing procedure in subspace at any level.

Algorithm 4 Give birth to a child

KNOWN: i, c_i^{sup}, c_i, T, C^{LB}, C^{UB}, $\mathbb{S}^{\text{LB}}$, $\mathbb{S}^{\text{UB}}$, C^{UBLR}; E (the error), $\mathcal{M}$ (the cumulative sum matrix)
CONSTRUCT: The child space object

```
1: $\triangleright$ Make a copy of the parent as the child. Do the followings in the child space.
2: $\triangleright$ Feed $T$, $C^{\text{LB}}$, $C^{\text{UB}}$, $\mathbb{S}^{\text{LB}}$, $\mathbb{S}^{\text{UB}}$, $E$ to the squeezing function. $C^{\text{LB}}$ and $C^{\text{UB}}$ are updated to $C^{\text{inf}}$ and $C^{\text{sup}}$
3: $i \leftarrow \arg\min_{i \in [1,\, L]} (c_i^{\text{UB}} - c_i^{\text{LB}})$ $\triangleleft$ least branches
4: $c_i \leftarrow c_i^{\text{LB}}$
5: $c_i^{\text{sup}} \leftarrow c_i^{\text{UB}}$ $\triangleleft$ fix $c_i$ to the start and memorize the end of the sequence
6: $\mathbb{S}^{\text{LB}} \leftarrow \mathbb{S}^{\text{LB}} - S(c_i^{\text{LB}})$
7: Erase $c_i^{\text{LB}}$ from $C^{\text{LB}}$
8: $\mathbb{S}^{\text{UB}} \leftarrow \mathbb{S}^{\text{UB}} - S(c_i^{\text{UB}})$
9: Erase $c_i^{\text{UB}}$ from $C^{\text{UB}}$
10: $T \leftarrow T - S(c_i)$
11: $C^{\text{UBLR}} \leftarrow c_1^{\text{sup}} : c_{i-1}^{\text{sup}}$ $\triangleleft$ usefulness comes in alg. 5
12: $\triangleright$ Since $c_i$ is fixed, the upper bounds should be refreshed as $\big(\min(c_i - (i-1),\, c_1^{\text{UB}}), \ldots, \min(c_i - 1,\, c_{i-1}^{\text{UB}}),\, c_{i+1}^{\text{UB}}, \ldots, c_L^{\text{UB}}\big)$. The following loop refreshes the upper bounds and $\mathbb{S}^{\text{UB}}$.
13: for $k$ in $1 : i-1$ do
14:     if $c_i - k < c_{i-k}^{\text{UB}}$ then $c_{i-k}^{\text{UB}} \leftarrow c_i - k$
15:     else break
16:     end if
17: end for
18: if $k > 1$ then
19:     $\mathbb{S}^{\text{UB}} \leftarrow \mathbb{S}^{\text{UB}} - \mathcal{M}(c_{i-k}^{\text{UB}} - 1,\, k-1) + \mathcal{M}(c_{i-k}^{\text{UB}},\, k-1)$
20: end if
```

VII. **Summary on FLSSS**

The above are the core ideas of FLSSS. In implementation, C++ STL vector is used as the fundamental data container. Originally, the index vectors store pointers to the elements in the superset. Considering most computers nowadays are 64–bit addressed, making a pointer 8 bytes, so in the current version, index vectors store various types of offsetting integers depending on the superset size. If superset size is less than 256, *unsigned char* is used; Else if it is less than 65536, *unsigned short* is used; Else *unsigned* is used. Compared to previous version, the current one has memory consumption decreased, but the speed is not found significant improved experiment.

A. *Conjugate Problem*

There exists a conjugate to every fixed size subset sum problem: given numeric vector $X = (x_1, x_2, \ldots, x_N)$, target $\sum X - T$, error E, find one or more subsets of X of size $N - L$ like $X_c = (x_{c_1}, x_{c_2}, \ldots, x_{c_{N-L}})$, such that $\sum_{i=1}^{N-L} x_{c_i} \in [\sum X - T - E, \sum X - T + E]$. Apparently, a fixed size subset sum problem and its conjugate are

Algorithm 5 Update space object

KNOWN: Current space object.
Update: The current space object

```
1:  if $c_i = c_i^{\sup}$ then
2:      return ▷ cannot be updated anymore
3:  end if
4:  ▷ We intend to increment $c_i$ by 1. This action needs refreshing the bounding vectors to follow up. First, the lower bounds should be refreshed as $\big(c_1^{\mathrm{LB}}, \ldots, c_{i-1}^{\mathrm{LB}}, \max(c_i + 1, c_{i+1}^{\mathrm{LB}}), \ldots, \max(c_i + L - i, c_L^{\mathrm{LB}})\big)$.
5:  $c_i \leftarrow c_i + 1$
6:  for $k$ in $1 : L - i$ do
7:      if $c_i + k > c_{i+k}^{\mathrm{LB}}$ then $c_{i+k}^{\mathrm{LB}} \leftarrow c_i + k$
8:      else break
9:      end if
10: end for
11: if $k > 1$ then
12:     $\mathbb{S}^{\mathrm{LB}} \leftarrow \mathbb{S}^{\mathrm{LB}} - \mathcal{M}(c_{i+1}^{\mathrm{LB}} - 1, k - 1) + \mathcal{M}(c_{i+1}^{\mathrm{LB}}, k - 1)$
13: end if
14: ▷ Second, the upper bounds should be refreshed as $\big(\min(c_i - (i - 1), c_1^{\mathrm{UBLR}}), \ldots, \min(c_i - 1, c_{i-1}^{\mathrm{UBLR}}), c_{i+1}^{\mathrm{UB}}, \ldots, c_L^{\mathrm{UB}}\big)$. We can see the use of $C^{\mathrm{UBLR}}$ is to prevent the upper bounds from being lifted up through their maximums — $C^{\sup}$ in algorithm 4. Refreshing the lower bounds does not need similar implementation is because as $c_i$ incremented, the lower bounds can only go up from the beginning when they were equal to the parent's lower bounds.
15: for $k$ in $1 : i - 1$ do
16:     if $c_i - k \le c_{i-k}^{\mathrm{UBLR}}$ then
17:         ▷ Be careful when $c_i - k = c_{i-k}^{\mathrm{UBLR}}$, we keep refreshing the left, since $c_i$ was incremented at line 5.
18:         $c_{i-k}^{\mathrm{UBLR}} \leftarrow c_i - k$
19:     else break
20:     end if
21: end for
22: if $k > 1$ then
23:     $\mathbb{S}^{\mathrm{UB}} \leftarrow \mathbb{S}^{\mathrm{UB}} - \mathcal{M}(c_{i-k}^{\mathrm{UB}} - 1, k - 1) + \mathcal{M}(c_{i-k}^{\mathrm{UB}}, k - 1)$
24: end if
```

equivalent. If one of the two finds a solution, then its complement will be a solution to the conjugate problem.

Results from experiments showed there usually exist a pronounced difference between the time costs of solving a fixed size subset sum problem and its conjugate, which is understandable: the pair of problems go through different searching trees.

Figure 4 shows the time costs (in seconds) of searching for 1,000 solutions where $X = (1^3, 2^3, \ldots, 99^3, 100^3)$, $L = 20$, $E = 0.1$ and T equals sum of 20 random elements in X. The hardware parameters are Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz and 16GB RAM. The compiler is gcc 4.9.3 embedded in Rtools.

Original	Conjugate
0.37	1.94
16.96	0.4
0.08	0.02
15.16	0.47
0.85	1.29

Fig. 4: 1000 solutions Time Cost (in seconds)

B. *General Subset Sum*

An obvious way of solving the subset sum in general (unfixed subset size) is looping over the subset sizes from 1 to N.

A more "mathematically elegant" way is to pad N 0s in the superset and to solve a size–N subset from the size–$2N$ superset. Once a solution is found, we can eliminate the indexes pointing to those redundant 0s and map the rest to the original superset. In the package, defining subset size 0 will note the function to run this method of solving the general subset sum problem. Details and examples can be found in the package manual.

This method would take larger memory and might easily increase the total time cost because of larger combinatorial space. However it could save user from the thinking of how to allot appropriate amount of individual running time for different subset sizes.

VIII. Multidimensional Fixed Size Subset Sum

Alright, let's be more ambitious. What if the superset is multivariate, say something like

$$\boldsymbol{X} = \{(1,\, 13,\, 208),\, (64,\, 3.4,\, 7),\, \ldots,\, (7,\, 9,\, 103)\}$$

and we want a subset of size 10 such that the sum of the elements falls in

$$[(436,\, 334,\, 354) - (1,\, 12,\, 8),\, (436,\, 334,\, 354) + (1,\, 12,\, 8)]$$

To analyze the problem, denote the d–variate superset vector by an $N \times d$ matrix

$$\begin{aligned}\boldsymbol{X} &= (X_1, \ldots, X_d)^T \\ &= (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) \\ &= \begin{pmatrix} x_{11} & \ldots & x_{1d} \\ \vdots & \vdots & \vdots \\ x_{N1} & \ldots & x_{Nd} \end{pmatrix}\end{aligned} \tag{10}$$

where $X_1, \ldots, X_d$ are column vectors and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ are row vectors. Additionally, denote the multivariate target by

$$T = (t_1, \ldots, t_d)^T \tag{11}$$

and the multivariate error by

$$E = (e_1, \ldots, e_d)^T \tag{12}$$

One may come up with the idea of extending all the arithmetic operators from single–dimension to multidimension and load them on the existing algorithm of FLSSS. For example, the less than operator for two d–variates $\boldsymbol{u}$ and $\boldsymbol{v}$ would be:

$$\boldsymbol{u} < \boldsymbol{v} := u_1 < v_1 \cap \ldots \cap u_d < v_d \tag{13}$$

and the additive operator would be:

$$\boldsymbol{u} + \boldsymbol{v} := (u_1 + v_1,\, \ldots,\, u_d + v_d) \tag{14}$$

However, to expect this to work, like the single–dimensional FLSSS, the superset must have:

$$\boldsymbol{x}_1 \leq \ldots \leq \boldsymbol{x}_N \tag{15}$$

In other words, the column vectors $(X_1,\, \ldots,\, X_d)$ must be perfectly rank–correlated — comonotonic, so that ineq. 15 is possibly achieved via sorting.

Are you frustrated to see the way is blocked? I was. But later I found some trick to poke $\boldsymbol{X}$ around. Consider a simple example of a two–variate superset

$$\boldsymbol{X} = \begin{pmatrix} -1 & 5 \\ 2 & 1 \\ 3 & 4 \end{pmatrix}$$

and assume an L–size subset sum problem about $\boldsymbol{X}$ has

$$T = (t_1,\, t_2);\; E = (e_1 = 0,\, e_2)$$

There is no way of sorting $\boldsymbol{X}$ to satisfy ineq. 15. However, if we scale the first column by a certain amount and add it to the second, like

$$\boldsymbol{X}^* = \begin{pmatrix} -1 & 5 + (-1) \times 10 = -5 \\ 2 & 1 + 2 \times 10 = 21 \\ 3 & 4 + 3 \times 10 = 34 \end{pmatrix}$$

then the second column becomes sorted. More importantly, we can be sure any solution to an L–size subset sum problem about $\boldsymbol{X}^*$ with

$$T^* = (t_1,\, t_2 + 10t_1);\; E^* = (e_1 = 0,\, e_2)$$

is also a solution to the original problem about $\boldsymbol{X}$. The proof is trivial.

A. *Dimension Shadowing*

Let's call the idea described above "dimension shadowing". We will study the possibility of using one dimension (key column) to shadow the rest.

The key column and any other constant /variable /vector /matrix that is shadowed by it will be superscripted with a star symbol *.

Consider the following superset

$$\begin{aligned} \boldsymbol{X} &= (X^*,\, X_2,\, \ldots,\, X_d)^T \\ &= \begin{pmatrix} x_1^* & x_{12} & \ldots & x_{1d} \\ \vdots & \vdots & \vdots & \vdots \\ x_N^* & x_{N2} & \ldots & x_{Nd} \end{pmatrix} \end{aligned} \tag{16}$$

where the rows have been ordered by the key (which column should be chosen as the key will be discussed later) — the first column. To shadow a certain column X_i $(2 \leq i \leq d)$, we add the scaled vector $\alpha_i X^*$ where $\alpha_i \geq 0$ to X_i such that

$$X_i^* = \alpha_i X^* + X_i \tag{17}$$

in which the elements satisfy

$$x_{1i}^* \leq x_{2i}^* \leq \, \ldots \, \leq x_{Ni}^* \tag{18}$$

To find α_i, define $\mathbf{d}$ as the vector differential operator, so ineq. 18 is equivalent to

$$\begin{aligned} &\mathbf{d}(\alpha_i X^* + X_i) = \mathbf{d}X_i^* \\ &= (x_{2i}^* - x_{1i}^*,\, x_{3i}^* - x_{2i}^*,\, \ldots,\, x_{Ni}^* - x_{(N-1)i}^*) \geq \mathbf{0} \end{aligned} \tag{19}$$

Thus α_i must satisfy

$$\begin{aligned} &\Big(\bigcap_{k=2}^{N} \alpha_i(x_k^* - x_{k-1}^*) + x_{ki} - x_{(k-1)i} \geq 0\Big) \cap \alpha_i \geq 0 \\ \iff &\Big(\bigcap_{k=2}^{N} \alpha_i \geq -\frac{x_{ki} - x_{(k-1)i}}{x_k^* - x_{k-1}^*}\Big) \cap \alpha_i \geq 0 \\ \iff &\alpha_i = \max\Big(-\frac{\mathbf{d}X_i}{\mathbf{d}X^*},\, 0\Big) \end{aligned} \tag{20}$$

Apparently, to make it work, none of the elements in $\mathbf{d}X^*$ can equal 0. In other words,

$$\text{The key column must be a strictly increasing vector} \tag{21}$$

Let $[t^{*-},\, t^{*+}]$ be the subset sum range for the key column, $[t_i^-,\, t_i^+]$ be the subset sum range for X_i. We need to calculate the appropriate subset sum range $[t_i^{*-},\, t_i^{*+}]$ for X_i^*.

Assume the key meets requirement 21. Suppose after all the rest dimensions are shadowed and running mFLSSS (FLSSS extended to multidimensional space) on $\boldsymbol{X}^*$ provides a solution, then the solution must satisfy:

$$\begin{aligned} t^{*-} &\leq s^* \leq t^{*+} \\ t_i^{*-} &\leq s_i^* \leq t_i^{*+} \end{aligned} \tag{22}$$

where

- s^* is the subset sum of the solution's key column.
- s_i^* is the subset sum of the solution's ith column.

Based on eq. 17, sum of the pre–shadowed ith column of the solution would be

$$s_i = s_i^* - \alpha_i s^* \tag{23}$$

And from eq. 22 we get the bounds for s_i:

$$t_i^{*-} - \alpha_i t^{*+} \leq s_i \leq t_i^{*+} - \alpha_i t^{*-} \tag{24}$$

Equalize the bounds to $s_i \in [t_i^-,\, t_i^+]$, we can solve t_i^{*-} and t_i^{*+}:

$$[t_i^{*-},\, t_i^{*+}] = [t_i^- + \alpha_i t^{*+},\, t_i^+ + \alpha_i t^{*-}] \tag{25}$$

Here comes a problem: to make eq. 25 legit, the upper bound must be no less than the lower bound, that is

$$t_i^- + \alpha_i t^{*+} \leq t_i^+ + \alpha_i t^{*-} \iff \alpha_i \leq \frac{t_i^+ - t_i^-}{t^{*+} - t^{*-}} \tag{26}$$

However, α_i is calculated from eq. 20. We have no guarantee of ineq. 26 being true.

B. *Key Dimension Adjustment*

In the best scenario, after calculating α_i, $i \in [2, d]$, none of them fails ineq. 26. Then we can compute $[t_i^-, t_i^+]$ and load mFLSSS. But the world is not perfect. We must think of a backup plan.

Observing ineq. 26, the upper bound of α_i is controlled by the key column subset sum range width $t^{*+} - t^{*-}$. If it is infinitely small, or just 0, then α_i is bounded by nothing, and ineq. 25 would always be legit.

Holding this idea, how about we adding a redundant column as the key dimension in which the elements are an integer sequence? Summing integers will always be an integer, so the subset sum range width can be set to 0.

Add the key column $1 : N$ to $\boldsymbol{X}$:

$$\begin{aligned}\boldsymbol{X} &= (X^*, X_1, \ldots, X_d)^T \\ &= \begin{pmatrix} 1 & x_{11} & \ldots & x_{1d} \\ \vdots & \vdots & \vdots & \vdots \\ N & x_{N1} & \ldots & x_{Nd} \end{pmatrix}\end{aligned} \tag{27}$$

Then eq. 25 becomes

$$[t_i^{*-}, t_i^{*+}] = [t_i^-, t_i^+] + \alpha_i t^* \tag{28}$$

where t^* is the key column subset sum. No bounding issue will happen to α_i any more.

Yet we have a new problem: what is the value of t^*? The original $\boldsymbol{X}$ does not have this redundant column, thus no requirement on the subset sum on this dimension. So t^* could be any value.

Fortunately, because X^* is an integer sequence, summing up any L (subset size) of them can be no less than $\sum_{k=1}^{L} k$ and no greater than $\sum_{k=N-L+1}^{N} k$. Therefore, all the possible subset sums for the key column would be an integer sequence of size $L(N-L)+1$:

$$\begin{aligned} t^* &\in \Big(\sum_{k=1}^{L} k\Big) : \Big(\sum_{k=N-L+1}^{N} k\Big) \\ &= \frac{1}{2}L(1+L) : \frac{1}{2}L(2N-L+1) \end{aligned} \tag{29}$$

This implies that to finally solve the problem, we could kick off $L(N-L)+1$ mFLSSS function calls, each of which has a different key column subset sum in sequence 29 with 0 error.

A better way is to design a non–zero error so the number of mFLSSS calls may be reduced. More specifically, from ineq. 26 we know the range of key column subset sum must satisfy:

$$t^{*+} - t^{*-} \le \frac{t_i^+ - t_i^-}{\alpha_i} \tag{30}$$

Therefore, let

$$2e^* = t^{*+} - t^{*-} = \min\big(\frac{t_1^+ - t_1^-}{\alpha_1}, \ldots, \frac{t_d^+ - t_d^-}{\alpha_d}\big) \tag{31}$$

If $2e^*$ is less than 1, it means the subset sum range is not wide enough to cover more than one integer in an consecutive integer sequence, then we stay with the original method of calling mFLSSS $L(N-L)+1$ times. Otherwise, the total number of mFLSSS calls can be reduced to

$$\lceil \frac{L(N-L)}{2e^*} \rceil \tag{32}$$

with key column subset sum ranges:

$$\begin{aligned} &\Big[\frac{1}{2}(1+L)L,\ \lfloor \frac{1}{2}(1+L)L + 2e^* \rfloor\Big], \\ &\Big[\lceil \frac{1}{2}(1+L)L + 2e^* \rceil,\ \lfloor \frac{1}{2}(1+L)L + 4e^* \rfloor\Big], \\ &\Big[\lceil \frac{1}{2}(1+L)L + 4e^* \rceil,\ \lfloor \frac{1}{2}(1+L)L + 6e^* \rfloor\Big], \\ &\qquad \vdots \end{aligned} \tag{33}$$

C. *Algorithm*

Algorithm 6 is a summary of the multidimensional FLSSS implementation.

The **for** loop at line 16 and 22 can be parallelized. Several earlier versions of FLSSS simply randomly sampled $\frac{(N-L)L+1}{D}$ (D being the number of threads) or $\frac{\lceil L(N-L)/2e^* \rceil}{D}$ elements from the subset sum sequence for each thread. However, because the time mFLSSS would take for a certain β is unpredictable, it is often the case that some threads quit too early. This problem has been resolved in the current version by enforcing communication among threads to dynamically control the number of βs each thread will work on.

D. *Conjugate Pair*

Difference in the time costs of solving a multidimensional fixed size subset sum problem and its conjugate is more pronounced than that for FLSSS, which was discussed in section VII. It is recommended to try both simultaneously. However, if computing resource allows, spawning as many as possible threads is still the best way to find the required number of solutions in the shortest time. Such time cost is not linear to the number of threads invoked, but usually, decreases much faster as the threads grow.

IX. Multi–objective Multidimensional Knapsack Problem

One of the main values of the Multidimensional Fixed Size Subset Sum Solver (MFSSSS) is that it can be applied to solve a general–purpose Knapsack Problem. To MFSSSS, there is no difference among whether the knapsack problem is multi–objective or multidimensional or both.

Consider the following $N \times d$ multidimensional multi-objective knapsack problem:

$$\boldsymbol{X} = (X_1, X_2, \ldots, X_V, X_{V+1}, \ldots, X_d) \tag{34}$$

Algorithm 6 Multidimensional FLSSS

```
KNOWN:
  1) Matrix $X = [X_1, \ldots, X_d]$ where $X_i$ is the $i$th column.
  2) Subset size $L$.
  3) Subset sum bounds. $(t_1^-, \ldots, t_d^-)$ and $(t_1^+, \ldots, t_d^+)$.
  4) mFLSSS — the multivariate version of FLSSS.
SOLVE: Multidimensional Fixed Size Subset Sum Problem
 1: Calculate the rank–correlations among columns in $X$. Denote the correlation matrix by $R_X$.
 2: For $R_X$, calculate the column sum or row sum vector $S_{R_X}$. Denote the $i$th element by $S_{R_X}(i)$.
 3: if $\sum_{i=1}^{d} S_{R_X}(i) = d^2$ then
 4:     Load mFLSSS and return. ▷ The condition means all the columns are perfectly rank correlated.
 5: end if
 6: Select $i \in 1 : d$ where $S_{R_X}(i)$ is the largest as the key. Switch the key with the first column in $X$, which then becomes $[X^*, X_2, \ldots, X_d]$. If $\mathbf{d}X^*$ contains 0, find the next largest $S_{R_X}(i)$ and select the corresponding $X_i$ as the key column. Keep doing such until the differential of the key contains no 0. If no such column found, head to line 13.
 7: Calculate $\alpha_i$ for all $i \in 2 : d$ based on eq. 20.
 8: if $\alpha_i$ satisfy eq. 26 for all $i \in 2 : d$ then
 9:     Shadow $X_2, \ldots, X_d$ according to eq. 17
10:     Compute $[t_i^{*-}, t_i^{*+}]$ for all $i \in 1 : d$ from eq. 25.
11:     Load mFLSSS on the shadowed matrix and return.
12: end if
13: Add integer key column $1 : N$ to $X$, as in eq. 27.
14: Calculate $\alpha_i$ for all $i \in 1 : d$, shadow the matrix and subset sum ranges.
15: if $2e^* < 1$ in eq. 31 then
16:     for $\beta$ in $\frac{L}{2}(1 + L) : \frac{L}{2}(2N - L + 1)$ do
17:         Compute $[t_i^{*-}, t_i^{*+}]$ from eq. 28 for all $i \in 1 : d$.
18:         Set subset sum range for the key as $[\beta - \epsilon, \beta + \epsilon]$ where $0 < \epsilon < 1$.
19:         Load mFLSSS on the shadowed matrix with the new target range.
20:     end for ▷ This loop is parallelizable.
21: else
22:     for $\beta$ in $1 : \lceil \frac{L(N-L)}{2e^*} \rceil$ do
23:         Set the key column subset sum range to the $i$th element in sequence 33.
24:         Load mFLSSS on the shadowed matrix with the new subset sum range.
25:     end for
26: end if
27: Collect the solutions and return.
```

where

- $X_1, \ldots, X_V$ are the value dimensions. A knapsack problem asks for a subset of the N objects such that the sum among these dimensions are as large as possible.
- $X_{V+1}, \ldots, X_d$ are the capacity dimensions. A knapsack problem would require a subset of the N objects to sum less than a value or within a certain range among these dimensions.

If we fix the subset size to L, this knapsack problem essentially becomes a multidimensional fixed size subset sum problem. Several aspects can be noted:

- Any size–L subset sum will be no greater than sum of the L largest elements and no less than sum of the L smallest elements in the superset, so any dimension is potentially bounded from two sides. For this reason, to maximize the value dimensions, one can first set the lower bound of the subset sum range close to sum of the L largest elements in superset. If it fails/succeed for a solution, lower/raise the bound and try again. The trial and error procedure can be automated, and practically, a heuristic solution — where the subset sum is close enough to the maximum — is often satisfactory.
- Minimizing a value dimension is equivalent to maximizing its negative.
- For any of the weight dimensions, if it is bounded only from one side, set the bound on the other side to sum of the L largest/smallest elements in superset.

For the unfixed general–purpose knapsack problem, we can apply the zero–padding method discussed in section VII-B. Practical examples can be found in the package manual.

X. Summary

So that is how long FLSSS have gone so far. I hope more advancements can be made in future, for example, if only we can mine some information about the bounding vectors of the old child for those of the new one to converge faster in the squeezing procedure. I have some intuitions but they seem not to work based on experiments. I believe the only way to confirm the road is dead–end or the opposite is to dig the math more rigorously.

Many people nowadays are fascinated about applying machine learning algorithms or Mont Carlo–based methods, which are usually inherently slow, to combinatoric problems. One of the most recent papers [2] covered some work on using recurrent neural network for the subset sum problem. However it only gives benchmarks on a naive example and no concrete code is provided either. My colleague once took months to run a relatively small asset management optimization with genetic algorithm. I think people like these algorithms is because they are in tread, mathematically elegant, and can provide a rich research resource. Additionally, they usually do not require high programming skills. However, making good efforts in thinking of classical non–statistical based algorithms is still worthy. FLSSS is probably a good example.

References

[1] Dirk Eddelbuettel, Romain Francois, JJ Allaire, Kevin Ushey, Qiang Kou, Nathan Russell, Douglas Bates, John Chambers. *Seamless R and C++ Integration*. The Comprehensive R Archive Network, Nov. 2016.

[2] Shenshen Gu, Rui Cui. *A finite-time convergent recurrent neural network based algorithm for the L smallest k-subsets sum problem*. Advances in Neural Networks ISNN 2013, Volume 7951 of the series Lecture Notes in Computer Science pp 19-27.

[3] JJ Allaire, Romain Francois, Kevin Ushey, Gregory Vandenbrouck, Marcus Geelnard, RStudio, Intel, Microsoft. *Parallel Programming Tools for Rcpp*. The Comprehensive R Archive Network, Aug. 2016.

[4] JJ Allaire, Hadley Wickham and others. *Rstudio*. RStudio, Inc, Dec. 2010–Current.

[5] Henrik Bengtsson. *R.rsp: Dynamic Generation of Scientific Reports*. The Comprehensive R Archive Network, Dec. 2016.