Building Blocks and Excluded Sums

By Erik D. Demaine, Martin L. Demaine, Alan Edelman, Charles E. Leiserson, and Per-Olof Persson

If we peel away, layer by layer, the complexities of the fast multipole method, we find that its inner core is a computation of:

Excluded sums: $\sum_{j \neq i} x_j$ and $\sum_{|j-i|>1} x_j$

In this model we are adding numbers x_j and excluding a neighborhood of x_i. In the FMM, the x_j become representations of functions, which are accurate only at some distance from point i.

This core, though perhaps obvious, was buried for many years. It took a trip to Japan and years of classroom presentations (Edelman, Leiserson), and a recent conversation over lunch at MIT (Demaine, Demaine, Edelman, Persson) before we could articulate the essence of the FMM. Our hope is that this distillation will connect people working on N-body problems to geometric algorithms. The goal is a stable computation of all N excluded sums with $\mathcal{O}(1)$ work per sum.

An obvious approach—to add all the x_j and subtract the excluded x_i ("`sum(x)-x`" in MATLAB)—fails for functions of interest, because we would be adding and subtracting singularities. This is like the cancellation that occurs in floating-point subtraction. Thus, we require that the excluded sums be computed *without subtractions*!

In one dimension, the row of numbers x_j can be added pairwise. Applied recursively, this rule yields the overall sum. Going "down the tree," we add parents and neighbors such that each level contains the excluded sum. The method requires $\mathcal{O}(N)$ additions and storage for n elements, and it generalizes to higher dimensions with the addition of blocks. This is essentially the *parallel prefix* algorithm for computing sums on parallel computers. It underlies the usual fast multipole method, in which the tree structure guarantees accuracy. The tree algorithm is shown in Figure 1, with excluded sums computed for $x_i = i$.

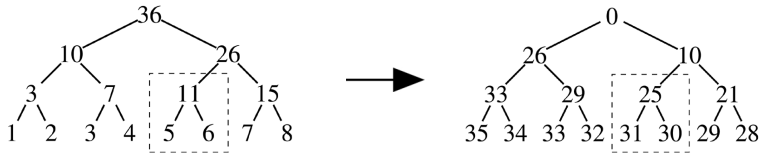

Figure 1. *The tree algorithm for input data 1, . . . ,8. It is easy to follow the pairs up the tree (left). Moving down the tree (right), siblings are swapped and added to the new values of their parents; for example, siblings 5 and 6 (lefthand box) are swapped and then added to their new parent, 25 (righthand box). The last row shows the sums with 1, . . . ,8 excluded.*

An easier and more intuitive approach abandons the tree structure. First, as shown in Figure 2, we add the x_j starting from the left, to obtain the prefix sum P_i ("`cumsum(x)`"). We compute the suffix sum S_i in the same way, but from the right. The excluded sum is $y_i = P_{i-1} + S_{i+1}$, and the neighborhood exclude is $y_i = P_{i-2} + S_{i+2}$.

Input data		1	2	3	4	5	6	7	8	
Prefix sums P_i			1	3	6	10	15	21	28	36 →
Suffix sums S_i	36	35	33	30	26	21	15	8		←
Excluded sums		35	34	33	32	31	30	29	28	

Figure 2. *The prefix algorithm. The prefix and suffix sums P_i,S_i are computed as cumulative sums starting from the left and from the right. The excluded sum is then $P_{i-1} + S_{i+1}$.*

Compared with the tree algorithm, the prefix algorithm is easier both to implement and to understand. It has fewer total operations for neighborhood exclude (true also in higher dimensions), and does not require that $n = 2^k$. It forms the basis for a new version of the fast multipole method now being developed (Edelman, Persson), with variation also in the function representations.

Can the algorithm be generalized to higher dimensions? For the excluded sum, we can treat the data as one long vector and stay in one dimension. But the neighborhood exclude needs a more sophisticated approach.

In two dimensions, there are four sums: Prefix sums for all the rows and then all the columns produce the prefix–prefix sum PP_{ij}. Similarly, we obtain PS_{ij}, SP_{ij}, and SS_{ij}, each $\mathcal{O}(N)$ to compute. What we need now is a geometric construct that will combine these sums in a nonoverlapping way to fill a rectangular domain, excluding a square in the middle. One way to do this is shown in Figure 3. The neighborhood of i,j is excluded in $PP_{i-2,j+1} + PS_{i+1,j+2} + SS_{i+2,j-1} + SP_{i-1,j-2}$.

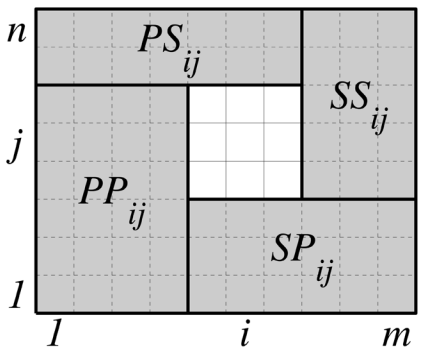

Figure 3. *The four terms PP_{ij}, PS_{ij}, SP_{ij}, and SS_{ij} "wrap around" the excluded center cell i,j to cover the shaded region.*

In D dimensions, we can create 2^D combinations of prefix and suffix sums. How do 2^D blocks from the corners combine to fill the original block, excluding an interior block? That is the central question considered in this article: Find 2^D blocks such that (1) they fill a D-dimensional $3 \times 3 \times \ldots \times 3$ block, except the center cube, and (2) each shares

a corner with the containing block. Some experimentation with building blocks produces an answer in three dimensions (see Figure 4).

For help in moving beyond three dimensions (and understanding the higher-dimensional problem) we (Edelman, Persson) described the problem over lunch to Erik and Martin Demaine. Shortly afterward, the e-mail exchange reproduced on the bottom of this page took place.

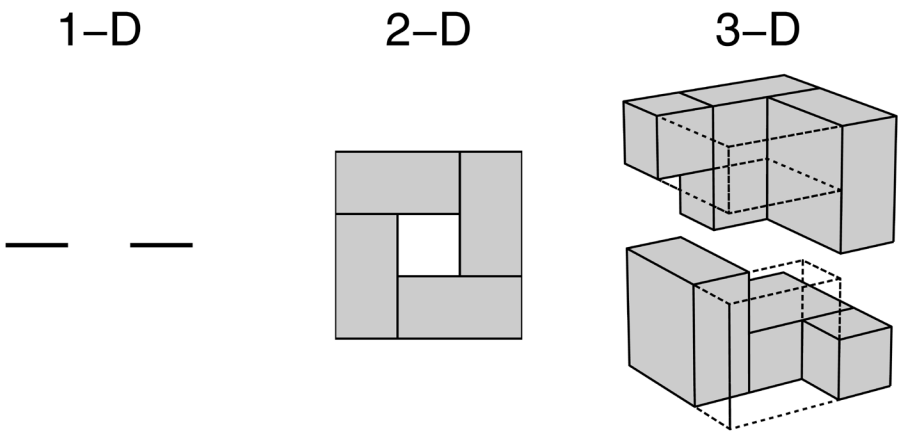

Figure 4. *Block configurations that exclude the center blocks in one, two, and three dimensions.*

From: edelman@math.mit.edu
|To: Erik Demaine <edemaine@mit.edu>
Date: Tue, 15 Jul 2003 12:46:57 -0400 (EDT)
Subject: Re: 3d geometry

Can you do this in arbitrary dimensions?

From: Erik Demaine <edemaine@mit.edu>
To: Alan Edelman <edelman@math.mit.edu>
Cc: Martin Demaine <mdemaine@mit.edu>
Date: Tue, 15 Jul 2003 16:52:39 -0400 (Eastern Standard Time)
Subject: Re: 3d geometry

Yes. Nice question. It led us to discover a general construction for such beasts. The construction is recursive, and behaves differently depending on the parity of the dimension.

Let's start with 1-D. Here there are only two unfilled squares surrounding the hole:

```
? ?
```

We can assign them to boxes arbitrarily:

```
A B
```

Let's move to 2-D. We know that every one-dimensional cross-section of a 2-D solution must be an instance of the 1-D solution. Hence we obtain the following information:

```
?A?
D B
?C?
```

There are now four letters, and each one needs to extend so that it touches a corner. There are also exactly four unmarked corners. What is essential for this to work is that the connections between unmarked corners and singleton letters that need to be extended form a cycle. Therefore, we can choose one orientation of the cycle, and extend each letter into the next corner along the cycle. Thus:

```
DAA
D B
CCB
```

In 3-D, again we know that every two-dimensional cross-section of a 3-D solution must be an instance of this 2-D solution. This tells us a lot. The first cross-section tells us:

```
???  DAA  ???
???  D B  ???
???  CCB  ???
```

The second cross-section tells us:

```
???  DAA  ???
DEE  D B  FFB
???  CCB  ???
```

The third cross-section tells us:

```
?A?  DAA  ?F?
DEE  D B  FFB
?E?  CCB  ?C?
```

In fact, each of these cross-sections has a binary choice about whether it is clockwise or counterclockwise, but it does not matter which we choose. Every letter used so far has been used three times, but must be used a fourth time to form a box, as well as to grab a corner of the entire cube. This can be done in only one way:

```
DAA  DAA  FF?
DEE  D B  FFB
?EE  CCB  CCB
```

We are left with two question marks, which are at (diagonally opposite) corners, so they can simply be assigned their own letter each.

```
DAA  DAA  FFH
DEE  D B  FFB
GEE  CCB  CCB
```

Note how the 3-D case behaves identically to the 1-D case: after we derive all possible information from lower dimensions, there are exactly two unlabeled corners, and we can label them arbitrarily.

To solve the 4-D problem, again we know that every three-dimensional cross-section of a 4-D solution must be an instance of this 3-D solution. The first cross-section tells us:

```
???  ???  ???
???  ???  ???
???  ???  ???

DAA  DAA  FFH
DEE  D B  FFB
GEE  CCB  CCB

???  ???  ???
???  ???  ???
???  ???  ???
```

The second cross-section tells us:

```
???  DAA  ???
???  DII  ???
???  KII  ???

DAA  DAA  FFH
DEE  D B  FFB
GEE  CCB  CCB

???  JJL  ???
???  JJB  ???
???  CCB  ???
```

The third cross-section tells us:

```
???  DAA  ???
DEE  DII  NII
???  KII  ???
```

```
DAA  DAA  FFH
DEE  D B  FFB
GEE  CCB  CCB

???  JJL  ???
JJM  JJB  FFB
???  CCB  ???
```

The fourth cross-section tells us:

```
?A?  DAA  ?O?
DEE  DII  NII
?E?  KII  ?I?

DAA  DAA  FFH
DEE  D B  FFB
GEE  CCB  CCB

?J?  JJL  ?F?
JJM  JJB  FFB
?P?  CCB  ?C?
```

Now we have used all information available from lower dimensions, and every position is labeled except for the 16 corners. Also, there are exactly 16 letters, each of which must be assigned a corner. Some of these assignments are forced in order for the letters to form boxes:

```
DAA  DAA  ?O?
DEE  DII  NII
?EE  KII  ?II

DAA  DAA  FFH
DEE  D B  FFB
GEE  CCB  CCB

JJ?  JJL  FF?
JJM  JJB  FFB
?P?  CCB  CCB
```

Now all remaining letters that have not yet been assigned a corner are singleton letters (each appears only once). Furthermore, the connections between singleton letters and unassigned corners form a cycle, so we can choose an arbitrary orientation of the cycle, and assign each letter to the next corner along the cycle. For example:

```
DAA  DAA  NOO
DEE  DII  NII
GEE  KII  KII

DAA  DAA  FFH
DEE  D B  FFB
GEE  CCB  CCB

JJL  JJL  FFH
JJM  JJB  FFB
PPM  CCB  CCB
```

Again, notice how the 4-D solution acts just like the 2-D solution.

Running this algorithm one more time for 5-D produces the following decomposition:

```
DAA DAA NOO  DAA DAA NOO  44V 1YY 1YY
DEE DQQ NQQ  DEE DII NII  44V 1II 1II
SEE SQQ 5QQ  GEE KII KII  GXX KII KII

DAA DAA WWH  DAA DAA FFH  RRV RRT FFT
DEE DQQ UQQ  DEE D B FFB  RRV RRB FFB
SEE SQQ UQQ  GEE CCB CCB  GXX CCB CCB
```

```
JJL JJL WWH   JJL JJL FFH   RR6 RRT FFT
JJ2 JJ2 U33   JJM JJB FFB   RRM RRB FFB
ZZ2 ZZ2 U33   PPM CCB CCB   PPM CCB CCB
```

(!)

What remains to be shown is that indeed every dimension acts this way: in odd dimensions, all but two opposite corners are filled; and in even dimensions, several corners are unfilled, but they are connected in a cycle with singleton letters. This is still a bit mysterious to me, but the fact that it works up to five dimensions is pretty convincing.

After the E-mail

We recently found a combinatorial construction of the boxes (see theorem and proof in the box on page 6).

We hope that readers will enjoy thinking about this problem (as we have!). Approximation theory issues remain to be worked out for the fast multipole application (adding functions with singularities, or their finite representations in the true algorithm).

These issues, which are at the core of the fast multipole method, are the subject of a forthcoming paper (Edelman, Persson).

The authors are all at MIT, where they are members of the Computer Science and Artificial Intelligence Laboratory (Demaine, Demaine, Edelman, Leiserson), the Department of Electrical Engineering and Computer Science (E. Demaine, Leiserson), and the Department of Mathematics (Edelman, Persson).

New Combinatorial Construction

Progress on the problem described in the article "Building Blocks and Excluded Sums" (see page 4) reached *SIAM News* at press time, in the form of the following theorem and proof:

Theorem. *If we remove the center unit cube from a d-dimensional cube of edge length 3, then the resulting structure can be decomposed into* 2^d *boxes with edge lengths in* $\{1,2\}$.

Proof. We label all the component cubies with a d-length word, $b_d \ldots b_2 b_1$, from the alphabet $\{-1,0,+1\}$, and observe that the corners are those cubies whose word is from the alphabet $\{-1,+1\}$. For every corner we circle those $+1$'s or -1's that are at the end of a run with an extra even/odd condition:

$\{i : b_i b_{i-1} = -1\} \cup \{1\}$ if $\{b_1 b_d = (-1)^d\}$.

Allowing the circled entries to be either the original value or 0 defines a box incident on the corner. (The odd/even condition guarantees that not all the b_i are circled, so that we exclude the center!)

We can verify that every cubie other than the center is placed in a unique box by replacing 0's with either $+1$ or -1 in the unique way consistent with the construction of the box. To do this, we replace a run of 0's with either $+-+-+-\ldots$ or $-+-+-+\ldots$ by looking at the non-zero to the right or using the end condition on the vertex that $b_1 b_d = (-1)^d$ if the rightmost bit is 0. □

Corollary. *There are* $2\binom{d}{k}$ *boxes of the form* $2^k 1^{d-k}$, *where k and d have opposite parity.*