The Well-Rounded Total[*]

Matthew Skala[†]
mskala@ansuz.sooke.bc.ca

March 25, 2017

Abstract

We discuss issues associated with reporting totals of rounded data values, as in financial statements, such that the reported values sum correctly to the reported totals with as little loss of accuracy as possible. We find that when the totals form a forest, it is always possible to make the totals add up with less than one unit of rounding error in every reported number; moreover, a $O(n)$ algorithm can find such a solution with the additional constraint that at least half the numbers are rounded to nearest, that is with at most $^1\!/_2$ unit of rounding error in half of the values. However, to guarantee that reported totals will add up, it may be necessary for the maximum rounding error of any reported number to be arbitrarily close to one unit. We also give a $O(n \log n)$ algorithm to find the rounding which minimizes the maximum rounding error of any reported number, when the totals form a forest. When the totals do not form a forest, it is not always possible to round every number in a report to within one unit and have all the rounded totals add up, and we show that the associated decision problem is $\mathcal{NP}$-complete.

1 Introduction

Last January, the Acme Novelty Company sold one widget and one gizmo, each priced at \$1.49, and reported it on their financial statement.

January Sales	
Widgets	\$1.49
Gizmos	\$1.49
Total	\$2.98

But the auditors complained that company policy required rounding amounts on financial statements to the nearest dollar.

[*]Preliminary draft. Check http://ansuz.sooke.bc.ca/entry/334 for updates.

[†]Funded by North Coast Synthesis Ltd., because nobody else would.

January Sales	
Widgets	\$1
Gizmos	\$1
Total	\$3

The value \$1.49 is nearer to \$1 than to \$2, so it is written as \$1. But the value \$2.98, sum of \$1.49 and \$1.49, is nearer to \$3 than to \$2. So by starting with the exact numbers down to the penny for both the line items and the total, and then rounding each number correctly to the nearest integer, the financial statement ended up saying that $1 + 1 = 3$, which is ridiculous.

The bookkeepers decided for February to use the floor function instead of rounding to nearest. That is, they would round each exact value to the next lesser integer value. That way, they thought, it wouldn't matter whether they did the addition before or after rounding: the floor of \$2.98 is \$2, which is also the sum of the floors of \$1.49 and \$1.49, so the January statement would have added up if that rule had been followed. It would enlarge the rounding error by a factor of 49 (from \$0.02 to \$0.98), but it would still be plausible that any number between \$2 and \$3 could round to \$2, and the auditors signed off on that as not a "material" problem as long as the rules to be followed were clearly documented.

In February they sold a widget and a gizmo, like in January, but also a doodad this month, each for \$1.49, and they applied the floor function according to the new policy, and again the numbers didn't add up.

February Sales	
Widgets	\$1
Gizmos	\$1
Doodads	\$1
Total	\$4

The floor function does not work. The ceiling function would not work either: it would round the \$1.49 values to \$2 each for a total of \$6, and round the exact total of \$4.47 only to \$5. Let a *well*-rounding rule be a rule that rounds any real number to either the floor or ceiling of that real. Let a *first-order* rounding rule simply be one that is a function: given a real input, it produces an integer that is to be the rounded value of that input, and the same input always gives the same output and the output depends on nothing else. This definition captures round down (floor), round up (ceiling), and most of the reasonable or popular rules for rounding to nearest, which differ from each other in how they handle exact half-integer values like \$1.50. All first-order well-rounding rules will necessarily round \$1.49 to \$1 or \$2, three copies of which will sum to \$3 or \$6, and then round \$4.47 to \$4 or \$5, resulting in a February statement that cannot possibly add up.

For March, they decided, hang it all, they would just *force* the statement to add up, by rounding the line items first by a round-to-nearest rule and then adding up the rounded values to get the totals, instead of computing exact totals and rounding those. But in March, the sales department was unusually successful due to a nationwide fad for Acme-style novelties, and they sold many more items: one each of a hundred different products for $1.49.

March Sales	
Widgets	$1
Wadgets	$1
Gizmos	$1
Doodads	$1
⋮	⋮
Boffs	$1
Total	$100

The exact total ought to have been $149.00, and the auditors weren't happy with rounding that to $100, not at all. So, after much discussion and debate, it was decided to change the policy, and report cents in the financial report instead of rounding to dollars.

March Sales	
Widgets	$1.49
Wadgets	$1.49
Gizmos	$1.49
Doodads	$1.49
⋮	⋮
Boffs	$1.49
Total	$149.00

The fad didn't last, and in April, they had to liquidate all the extra products they had manufactured. They dumped the products in a bin out front of the factory and put up a sign saying "Mix and match, any three for a dollar!" They only had one taker for the offer [6].

April Sales	
Widgets	$0.33
Wadgets	$0.33
Boffs	$0.33
Total	$1.00

That doesn't add up, either. A proposal was floated to keep the accounts in higher precision, such as tenths or hundredths of cents; but no finite number of decimal digits would be enough. The real problem, after all, was not precision of the *calculations*. It's easy for a human being or a computer using exact rational arithmetic to correctly determine that $^1/_3 + {}^1/_3 + {}^1/_3 = 1$. The real problem was presenting those exact calculations as decimal numbers on the report in a way that would add up.

One creative bookkeeper (and the Acme manager made a note to keep an eye on that one, since creativity is not always an asset for bookkeepers) suggested a way out of this mess: *higher-order* well-rounding rules. That is, instead of looking at only one input, the exact number to be rounded, let the rounding policy also consider context, such as whether it was operating on a total or a line item; hidden data, such as how it had already rounded other numbers on the statement; and maybe even randomness.

Bearing in mind that \$1.49 is only slightly closer to \$1 than to \$2, it really seems reasonable to round any single \$1.49 value either way; and it further seems reasonable that bad cases where rounding to nearest causes totals to go wrong will tend to have those totals made up of many such values that could go either way, allowing extra flexibility. If under generally-accepted accounting practices (GAAP) the bookkeepers would be allowed to round everything up, or everything down, then they ought also to be allowed to round some things up and other things down as long as that was done in a disciplined way; especially if doing so helped to provide a more honest and accurate presentation of the overall financial situation. Then maybe the flexibility so allowed could be used to make the totals come out right without the problems observed in March when they tried to mandate adding up the rounded values.

If they required that every amount on the statement should be rounded to the round dollar or cent from its exact value *either up or down*, but without making a specific requirement on which way, then every one of the previous statements could be made to obey the well-rounding constraint on both line items and totals, and add up exactly, in dollars or cents as desired.

January Sales		**January Sales**	
Widgets	\$1.49	Widgets	\$1
Gizmos	\$1.49	Gizmos	\$2
Total	\$2.98	**Total**	\$3

February Sales		**February Sales**	
Widgets	\$1.49	Widgets	\$1
Gizmos	\$1.49	Gizmos	\$2
Doodads	\$1.49	Doodads	\$1
Total	\$4.47	**Total**	\$4

March Sales		**March Sales**	
Widgets	\$1.49	Widgets	\$1
Wadgets	\$1.49	Wadgets	\$1
Gizmos	\$1.49	Gizmos	\$2
Doodads	\$1.49	Doodads	\$1
⋮	⋮	⋮	⋮
Boffs	\$1.49	Boffs	\$2
Total	\$149.00	**Total**	\$149

April Sales		**April Sales**	
Widgets	\$0.34	Widgets	\$1
Wadgets	\$0.33	Wadgets	\$0
Boffs	\$0.33	Boffs	\$0
Total	\$1.00	**Total**	\$1

Is it always possible to make this work? Can there be a disciplined, systematic policy to choose which numbers to round up and which to round down so as to make it work whenever possible and allow no unnecessary scope for cheating? Can we do better than allowing the maximum rounding error to be so big as a whole rounding unit? What can be said about these matters using the tools of computer science?

2 Well-rounding on forests

Consider the following problem.

Problem 1 (Well-rounding on forests)**.** Let T be a forest whose nodes are labelled with real numbers such that the label of each internal node is equal to the sum of the labels of its children. Leaf labels are unconstrained. Find a labelled forest $\hat{T}$, identical to T except that the labels on nodes of $\hat{T}$ are integers, still obeying the property that the label of each internal node is equal to the sum of the labels of its children, and for every node with label v in T and $\hat{v}$ in $\hat{T}$, $|v - \hat{v}|$, called the *rounding error* of the node, satisfies $|v - \hat{v}| < 1$.

This is a formalization of the well-rounding problem faced by the Acme bookkeepers in the introduction: choosing rounded values for the numbers in a report, some of which are totals of others, so that the rounded numbers will add up while still reflecting the exact values without too much rounding error. The requirement for T to be a forest means that any totals and subtotals are nested in an orderly fashion, with each number being included in at most one total at the next higher level. Totals on a financial statement, including all those in the introduction, typically have such structure.

Let an *exact* node in an instance of Problem 1 be one whose label is an integer, and an *inexact* node be one whose label is not an integer. Exact nodes necessarily round exactly to their unrounded labels with zero rounding error, in every solution, because no other integer is at distance less than 1.

The following lemma offers some good news: Problem 1 can always be solved, and there is an efficient algorithm. We describe the running time as $O(n)$ "arithmetic operations on node labels" in order to avoid a more or less irrelevant inquiry into the time bounds on real-number arithmetic. In practice, this algorithm would normally be implemented using fixed-point numbers of effectively constant width, and the constant-time assumption would be reasonable; even in a theoretical context, the question of how long it takes to do arithmetic on reals can be usefully separated from the algorithm that makes use of the arithmetic. Note that our constant-time arithmetic operations include floor and ceiling, which are not always included in constant-time arithmetic computational models.

Lemma 2. Every instance of Problem 1 has a solution, and there exists an algorithm to find a solution for any n-node forest using $O(n)$ arithmetic operations on node labels.

Proof. Consider a node x with label v in T, which will have label $\hat{v}$ in $\hat{T}$, and suppose we have chosen a rounding direction for it, either $\hat{v} = \lfloor v \rfloor$ or $\hat{v} = \lceil v \rceil$ (these might be the same, if x is exact). If x has no children, we simply use the chosen direction. If it has children, we must choose directions for them such that the labels will add up correctly. Consider what would happen if every child y of x, with labels w in T and $\hat{w}$ in $\hat{T}$, were rounded down: $\hat{w} = \lfloor w \rfloor$. Rounding a real down to its floor cannot increase it, and always results in an integer, so the sum of the children's rounded labels with all children rounded down is an integer less than or equal to v. Similarly, the sum of the children's rounded labels with all children rounded up, is be an integer greater than or equal to v.

If we start with all children rounded down and change them one by one to round up, then at each step we either leave the sum unchanged (for an exact child) or increase it by exactly one (for an inexact child) and in this process, we visit every integer from an integer that is less than or equal to v up to an integer that is greater than or equal to v, and so we visit both $\lfloor v \rfloor$ and $\lceil v \rceil$ along the way.

Therefore regardless of the choice of rounding direction for x, there exists a choice of rounding direction for all the children of x such that they will add up to the chosen rounded label $\hat{v}$. Moreover, we can find such a choice in linear time: first scan the children's labels and compute the sum of their floors, then subtract that from the desired $\hat{v}$ to find the number of inexact children which should be rounded up. All other inexact children round down; rounding direction makes no difference for exact children; there will necessarily exist enough inexact children to realize the desired number in each direction; and they can be given their directions in linear time too, by visiting them in arbitrary order and assigning directions greedily.

Once a child's rounding direction is decided, we recurse on it to compute rounding directions for any descendants it may have. The overall running time remains linear. ☐

3 Optimal well-rounding on forests

The previous section establishes that every instance of Problem 1 has a solution, and we can find one in linear time. But the solution thus found is only guaranteed to round every number to error less than one rounding unit. We might hope for a better limit on rounding error. Let us define an optimization version of the problem. This is exactly the same as the previous definition except that the maximum rounding error of any node is given the name δ and we ask for a solution that minimizes it.

Problem 3 (Optimal well-rounding on forests)**.** Let T be a forest whose nodes are labelled with real numbers such that the label of each internal node is equal to the sum of the labels of its children. Leaf labels are unconstrained. Find a labelled forest $\hat{T}$ and a real number δ, where $\hat{T}$ is identical to T except that the labels on nodes of $\hat{T}$ are integers, still obeying the property that the label of each internal node is equal to the sum of the labels of its children, for every node with label v in T and $\hat{v}$ in $\hat{T}$, $|v - \hat{v}| \leq \delta$, and δ is minimized.

This problem is also reasonably easy to solve, at the cost of a logarithmic factor in running time.

Theorem 4. There exists an algorithm to solve Problem 3 for any n-node forest using $O(n \log n)$ arithmetic operations on node labels.

Proof. First, note that $\delta < 1$ for all input, by Lemma 2, and is necessarily equal to $v - \lfloor v \rfloor$ or $\lceil v \rceil - v$ for some label v on a node in the input forest. Therefore, on a forest with n nodes, δ takes one of at most $2n$ distinct values, and we can find them all and sort them in $O(n \log n)$ time.

Although this does not change the asymptotics, we can in fact limit it to $n + 1$ distinct candidate δ values. Each node can be rounded in two directions, for two values of the rounding error at that node, equal in case of a node whose unrounded value is an exact half-integer. If we take the best value of rounding error at every node (that is, round to nearest), the maximum rounding error overall can still be no less than the greatest of these. That is one candidate for δ. Then n more candidates correspond to the error from the less desirable rounding direction for each of the n nodes, any of which might (until we prove otherwise) turn out to be the node that determines the overall maximum rounding error.

Now, we describe an algorithm that determines whether it is possible to assign rounding directions to all nodes in the input forest such that each node has rounding error at most d, where d will be a candidate for the value of δ given as input to the algorithm. The algorithm does a postorder traversal of the forest (that is, leaves before roots, trees that are disconnected from each other in arbitrary order), assigning values to two flags for each node, saying whether the node can be rounded up, and whether it can be rounded down. For leaves, these values are easily computed by checking whether $|v - \hat{v}| \leq d$ for each proposed value of $\hat{v}$. For internal nodes, given these flag values on all the children, we can compute the minimum and maximum possible sums of the rounded labels of the children, and check the floor and ceiling of the internal node's unrounded label against those bounds (and directly against d) to determine whether it is feasible for this internal node to be rounded down, up, neither, or both. At any point if we find that some node can be rounded neither down nor up, then we can terminate with failure. Otherwise, we finish with every node labelled with its options at the current value of d. This d-checking algorithm runs in linear time.

By binary search on the sorted candidates for δ, with a linear-time check at each step, we can find the minimum value of d for which every node has at least one option. That is δ, and computing it requires $O(\log n)$ iterations of

$O(n)$ time each, for an overall cost of $O(n \log n)$. Then in a final pass over the marked-up forest, we can assign rounding directions to any nodes that still have two options, in the same manner as the algorithm of Lemma 2.

The result is a rounded labelling that minimizes the maximum rounding error, solving Problem 3 in $O(n \log n)$ time. □

The procedure above makes the maximum rounding error as small as it can be for an instance, but does not say how small that is in general. We might hope that we could achieve a better limit than one unit; there seems to be a gap between the obvious bound that no rounding procedure can do better than error $1/2$ on input that contains half-integer values, and the bound of rounding error 1 from Lemma 2. The following result shows that in fact, no better constant guarantee is possible: there are instances of Problem 3 for which the best possible rounding error δ is arbitrarily close to 1. In other words, Lemma 2 is tight, at least for bounds that do not depend on n.

Theorem 5. For any ϵ satisfying $0 < \epsilon < 1$, there exists a T such that δ in the solution to Problem 3 satisfies $\delta \geq 1 - \epsilon$.

Proof. Let $k = \lceil 1/\epsilon \rceil$. The instance T consists of $k - 1$ leaf nodes labelled $1/k$, and a single root having them all as children, with label $(k-1)/k = 1 - (1/k)$. Treating T as an instance of Problem 1, that is, bounding the rounding error of each node to less than 1, implies that every node rounds to either 0 or 1.

If all leaves round to 0, then the rounding error of the root (which also rounds to 0) is $1 - (1/k)$. If one leaf rounds to 1, then so does the root, and the rounding error of that one leaf is $1 - (1/k)$. Two or more leaves cannot round to 1 because then the root would round illegally to 2 or more. Therefore the maximum rounding error over nodes in every solution is $1 - (1/k)$; that is the value of δ, and it satisfies $\delta \geq 1 - \epsilon$. □

If the rounding error of the worst-rounded number in a report may be arbitrarily close to 1, then an algorithm that optimizes the rounding error of that one number, possibly at the cost of making others worse than necessary, may not be preferable. For a real application, we might prefer to use round-to-nearest rules wherever possible, and look for a solution where very many numbers are rounded using round-to-nearest, even if it means some must be rounded with a little more error, up to a full rounding unit. The following result provides a solution that may be desirable for applications: a guarantee that totals will add up and *at least half* of all numbers in the report will be rounded by round-to-nearest. The rounded-to-nearest numbers have rounding error at most $1/2$, and (coupled with Lemma 2) it implies that the average rounding error will be less than $3/4$ even if the remaining numbers may be rounded with rounding error close to 1.

Theorem 6. For every instance of Problem 1 there exists a solution in which at least half the nodes are rounded to the nearest integer, that is with rounding error at most $1/2$. Then the *average* (not maximum) rounding error of all nodes is necessarily less than $3/4$. Furthermore, there exists an algorithm that can find such a solution for any n-node instance using $O(n)$ arithmetic operations on node labels.

Proof. Starting with an instance T, construct T' as follows: double all the node labels in T; solve Problem 1 on the resulting labelled forest by any convenient method, such as that of Lemma 2; then divide all the labels in the result by two and call that T'. The nodes in T' all have integer or half-integer labels, each differing by less than $1/2$ from the corresponding labels in T.

We will construct a graph G as follows. The vertices of G are exactly those nodes of T' that have half-integer labels. For each *family* in T' consisting of an internal node and its children, it is necessarily the case that an even number of nodes have half-integer labels, because the parent's label is the sum of the children's labels. Choose an arbitrary perfect matching among the nodes in the family that have half-integer labels, and add the edges of the perfect matching to G. The result is that G is a union of disjoint paths (possibly including edgeless paths of just one vertex each) covering all the vertices.

We will construct a solution $\hat{T}$ as follows. Nodes of T' that have integer labels will be labelled the same in $\hat{T}$. Nodes of T' that do not have integer labels (therefore half-integer labels, and vertices of G) may each be rounded up (to the ceiling of the label in T) or down (to the floor). When a parent and child in T' are adjacent in G, we require them to round in the same direction. When two siblings are adjacent in G, we require them to round in opposite directions. This leaves one free choice of rounding a node up or down in each component path of G. For each selection of the arbitrary choices in constructing G and $\hat{T}$, these rules produce a solution to T that satisfies the requirements of Problem 1, because $\hat{T}$ rounds off T' with maximum rounding error at most $1/2$, and T' in turn rounds off T with maximum rounding error less than $1/2$.

Now, for every component of G, some of the corresponding nodes in $\hat{T}$ may be rounded to nearest (with rounding error from T at most $1/2$) by the arbitrary choice of direction, and others may not. If we change our selection of one rounding direction in the path, all other rounding directions in the path are also required to change. Then the counts of how many are rounded to nearest and how many are not, switch places. One choice or the other necessarily results in at least half of the nodes in the path being rounded to nearest. By making such choices for all components of G, at least half of the nodes that had half-integer labels in T' will be rounded to nearest in $\hat{T}$. Any nodes that did not have half-integer labels in T' were already rounded to nearest in $\hat{T}$. Therefore we can find a $\hat{T}$ in which at least half of all nodes overall are rounded to nearest from T, meeting the requirements of the theorem.

Each step in this process can be done by a linear time algorithm: finding T', constructing G, choosing a direction for each path, and then forming the final $\hat{T}$. □

Figure 1 illustrates how the graph G in the proof of Theorem 6 works to preserve the correct addition of totals. When the first rounding pass creates T', it guarantees that each parent is labelled with the sum of its children. The paths

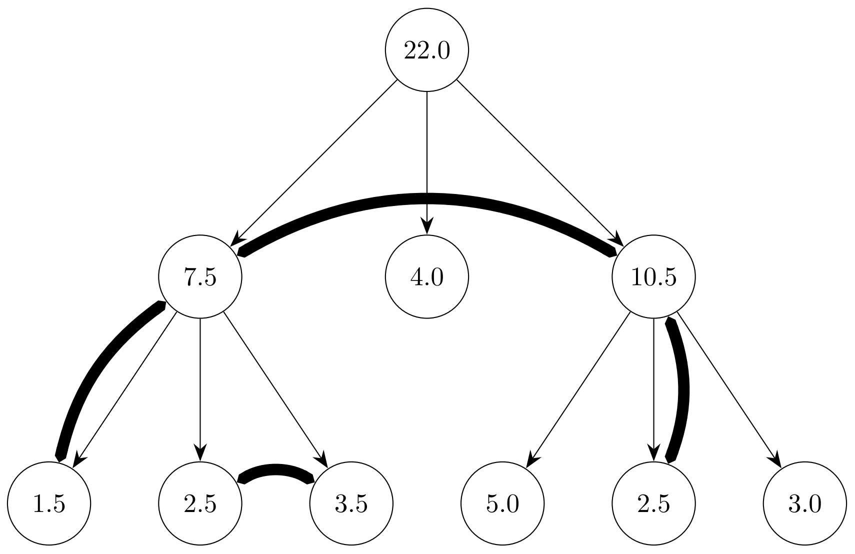

Figure 1: The forest T' with half-integer labels covered by the graph G, shown by the heavy lines.

in G provide sufficient conditions for further rounding the half-integer labels in T' to integers while keeping the totals correct. If two siblings are adjacent in G, like the 2.5-labelled and 3.5-labelled nodes in the lower left of the figure, then rounding one of them up and the other down leaves the sum unchanged. Our procedure for creating G ensures than in any family of a parent and its children, all the children with half-integer labels will be in such pairs—except possibly for one child that is instead paired with the parent. Then we can round that child, and the parent, both up or both down. Obeying these constraints is sufficient to ensure that the parent will remain labelled with the sum of its children's labels.

The proof of Theorem 6 leaves unspecified the choice of a perfect matching within each family to create G; it is shown that the algorithm can round at least half of the nodes to nearest regardless of how these perfect matchings are chosen. However, not all choices are equally good.

Suppose Figure 1 represents the top three levels of a ten-level tree. Regardless of the choices made constructing G there will necessarily exist a path in G that goes all the way from some leaf, up to the 7.5-labelled child of the root, then across to the 10.5-labelled child, and back down to another leaf, encompassing a total of 18 nodes; but exactly which nodes are on that path will depend very much on the perfect matching choices. If the fractional parts of the labels in the original T along this long path are uniformly distributed in the interval $(0, 1)$, then the choice of a rounding direction for one node, forcing the directions of all others in the path, can necessarily achieve at least nine of them rounded to nearest, but cannot be expected to do significantly better than that.

On the other hand, suppose the 7.5-labelled node had label 7.1 in the original T, and the 10.5-labelled node originally had label 10.9. If we could choose the path so that all the nodes below 7.1 along the path were also nodes whose preferred rounding direction was down, and all those below 10.9 along the path were nodes with preferred rounding direction up, then we could choose one rounding direction and follow the path constraint in such a way as to make them *all* round to nearest. And it seems reasonable that we might really be able to do that by careful heuristic selection of the perfect matchings that define G.

If we measure the quality of a solution by the number of nodes rounded to nearest, it seems clear that choosing good matchings for G can lead to better solutions. However, we have no proof that Theorem 6 can truly maximize the number of nodes rounded to nearest, even with the best possible G; and choosing the best possible G might be hard enough to impair the running time. If, for instance, we found we needed to compute minimum-weight perfect matchings for some weighting function on the complete graph of half-integer-labelled nodes within each family, we might be running algorithms superquadratic in number of vertices [1]. We now present an heuristic for creating G that although not proven optimal, at least can be run in linear time and reasonably expected to help in many practical cases. In the worst case, the guarantees of Theorem 6 will still hold.

We will choose perfect matching for the families in T' according to a postorder traversal, giving each node with a half-integer label one of four tags representing the preferred rounding direction of that node and any in the path built up below it:

- *heavy* if this node and all those in the path below it have preferred rounding direction of upward, that is, their labels in T are greater than or equal to their half-integer labels in T';
- *light* if this node and all those in the path below it have preferred rounding direction of downward, that is, their labels in T are less than or equal to their half-integer labels in T';
- *indifferent* for the special case where this node, and all those in the path below it, have exact half-integer labels in T as well at T', which means they can be optimally rounded in either direction; or
- *mixed* if none of these other three cases apply, which implies that rounding this path will necessarily result in rounding at least one node against its preferred direction.

The heuristic attempts to avoid creating paths that would prevent nodes from rounding to nearest. When it considers each family, it first attempts to join a heavy child with a light child, as long as both heavy and light children remain. Each such operation creates a final path in G along which all, not only half, of the nodes can be rounded to nearest.

When the supply of heavy or light children is exhausted, the heuristic joins the parent to a child, if the parent has a half-integer label in T' and needs to be joined to a child. If the parent's label in T is greater than its label in T', it is joined to a heavy child if possible; if less, a light child; otherwise, an indifferent child; if none of those rules joins it, then it will be joined to a mixed child; and as a last resort, the parent may be joined to any remaining child, even if opposite to its own preferred rounding direction. The parent's tag is set according to the definitions, for use later in the traversal.

After joining the parent as necessary, any remaining heavy or light children are joined to indifferent children wherever possible, and finally, any remaining children are joined arbitrarily. It is possible to follow these rules in linear time by scanning all the children in a family to put them in four lists, then working through the lists.

4 Well-rounding on dags

The previous sections describe techniques applicable to rounding totals that have tree or forest structure, with totals for categories possibly broken down into smaller sub-categories, through any number of levels, but without any totals overlapping. However, it is entirely possible that we might wish to have overlapping totals.

May Sales by Region

	Western	Eastern	**Total**
Widgets	\$1.48	\$1.03	\$2.51
Doodads	\$1.01	\$1.48	\$2.49
Total	\$2.49	\$2.51	\$5.00

If two totals overlap, without one being a subset of the other, then choosing how to round the numbers becomes more complicated. The model of totals and subtotals as nodes in a forest can easily be generalized to cover this situation by allowing the nodes to form a directed acyclic graph (dag) instead. But the existing algorithms depend on forest structure, that is, each node having at most one parent. And that is not only a technical requirement; it cuts to the heart of the rounding problem. If we choose to round a number in one direction to satisfy some total above it, that choice might make another total fail to add up. Overlapping totals depend on each other in ways that may turn out to be complicated.

Another issue may be apparent in the example: a typical presentation, if it includes overlapping totals, may end up showing two different derivations for a single higher-level total. The grand total is not the sum of the four subtotals for widgets, doodads, the Western region, and the Eastern region; if it were, it would be \$10.00, not \$5.00. Rather, it is the sum of the two subtotals for widgets and doodads, and independently, it is also the sum of the two subtotals for the Western and Eastern regions. What if those two sums have different answers because of rounding applied to the subtotals? Indeed, a similar situation on a balance sheet was part of the inspiration for the current work. We had total assets not matching total equity plus liabilities, despite each being correctly added at the top level, due to rounding issues within subtotals computed from a general ledger that did balance correctly.

In fact, when we enforce correct addition across the entire report, this issue disappears. As long as each raw number that is not a total appears only once in the dag, its rounding will be the same everywhere it is used in computing totals. Two different derivations for the same higher-level total will always work out to the same result when computed over the final rounded numbers. The situation of adding regional numbers, adding per-product numbers, and getting two different values for what should be the same grand total, cannot actually occur. When constructing the dag, we can add edges to express any one derivation of the grand total, simply leave out any other derivations, and be assured that they will still add up correctly.

The first questions to answer about making totals add up when they can overlap are whether it is always possible, and when it is possible, how difficult it may be. Is there an analog of Lemma 2? We begin by defining the well-rounding problem for directed acyclic graphs. This definition is more or less that of Problem 1, just with the terminology changed to refer to the more general structure. Since we will be using this definition primarily for an $\mathcal{NP}$-completeness proof, we set it up as a decision problem, and require the node labels to be rational numbers to simplify any questions about binary encoding.

Problem 7 (Well-rounding on dags)**.** Let T be a directed acyclic graph (dag) whose nodes are labelled with rational numbers such that the label of each node with out-degree greater than zero is equal to the sum of the labels of its successors. Sink labels are unconstrained. Accept if and only if there exists a labelled dag $\hat{T}$, identical to T except that the labels on nodes of $\hat{T}$ are integers, still obeying the property that the label of each node with out-degree greater than zero is equal to the sum of the labels of its successors, and for every node with label v in T and $\hat{v}$ in $\hat{T}$, $|v - \hat{v}|$ satisfies $|v - \hat{v}| < 1$.

The first important difference between the problem on dags and the problem on forests is that with dags it is *not* necessarily possible to make all totals add up while keeping all rounding errors less than one unit.

Theorem 8. There exist both yes-instances and no-instances of Problem 7.

Proof. Any instance of Problem 1 is a yes-instance of Problem 7 by Lemma 2. Now let T be a dag with four sinks, all labelled $^2\!/_5$; six sources each having a distinct pair of two of the sinks as successors (labelled $^4\!/_5$); and four sources each having a distinct set of three of the sinks as successors (labelled $^6\!/_5$). This dag is shown in Figure 2. It happens to be planar, and it corresponds to a financial report with four line items of \$0.40 each and all possible subtotals of two or three line items. We will show that this is a no-instance of Problem 7.

Suppose a $\hat{T}$ existed for this T satisfying the requirements of Problem 7. Each sink's rounded label in $\hat{T}$ is necessarily 0 or 1. Each source with two successors has an unrounded label of $^4\!/_5$ and also necessarily rounds to 0 or 1. Therefore no two sinks can round to 1; at most one sink can round to 1

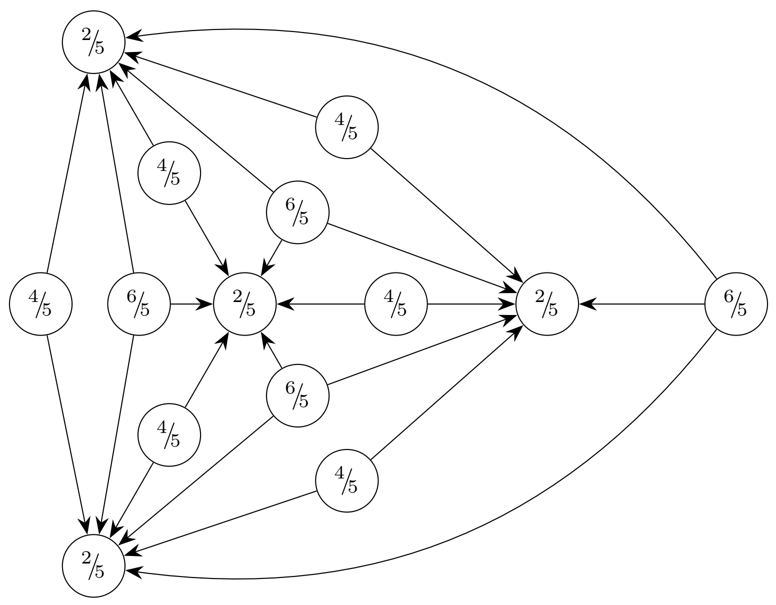

Figure 2: A no-instance of Problem 7.

overall, with the rest rounding to 0. But each source with three successors has an unrounded label of $^6\!/_5$, necessarily rounding to at least 1. If we choose at most one of the sinks to round to 1, there will be a source whose successors are exactly the other three sinks (or any three sinks if all were rounded to 0), all its successors will be rounded to 0, and that source's rounded label will necessarily be 0, giving a contradiction. Therefore $\hat{T}$ does not exist and T is a no-instance of Problem 7. □

Then the next question, if it does not work in all cases, is how to recognize the cases in which it works. The answer is disappointing, but unsurprising given that we have observed clues like long-range dependencies: Problem 7 is an $\mathcal{NP}$-complete decision problem. In general, if the totals on a financial statement are allowed to overlap, it may not only be impossible to make them all add up correctly when rounded, but also computationally infeasible to decide whether or not it is possible to make them all add up.

Theorem 9. Problem 7 is an $\mathcal{NP}$-complete decision problem.

Proof. Consider the one-in-three 3SAT problem limited to positive literals. That is, given a set of variables and a set of clauses which are triples of distinct variables, choose a value in $\{0, 1\}$ for every variable such that the sum of the variables in each clause is exactly 1. The one-in-three 3SAT problem limited to positive literals is known to be $\mathcal{NP}$-hard [4]. (The reference does not explicitly mention the limitation to positive literals, but gives a definition for the problem

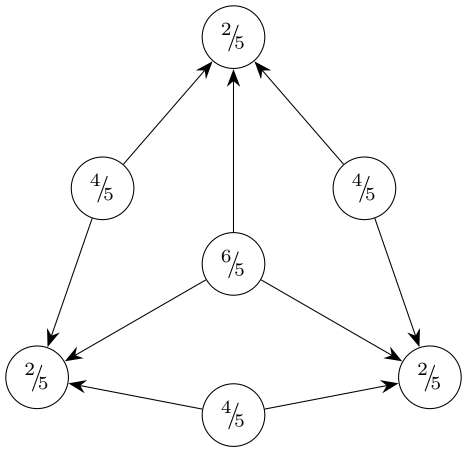

Figure 3: A gadget (not a widget or doodad) for clause satisfaction.

that implies this limitation; allowing *negated* literals is possible by an obvious reduction.)

Starting with any instance of the one-in-three 3SAT problem limited to positive literals, create a dag node labelled $^2\!/_5$ for each variable. For each pair of variables that appear together in some clause, create a node labelled $^4\!/_5$ with the nodes for those two variables as its successors. For each triple of variables that appear together as a clause, create a node labelled $^6\!/_5$ with the nodes for those three variables as its successors. The resulting gadget for clause satisfaction is as shown in Figure 3.

Well-rounded labelling of all seven nodes in this gadget, with correct totals, is possible if and only if exactly one of the three sinks is labelled 1 and the other two are labelled 0. Therefore a rounded labelling satisfying Problem 7 for this dag yields a solution to the original instance of one-in-three 3SAT limited to positive literals. Problem 7 is $\mathcal{NP}$-hard.

The rounded labelling is a polynomial certificate for a yes-instance; therefore Problem 7 is in $\mathcal{NP}$ and is $\mathcal{NP}$-complete. □

5 Discussion and conclusions

We are looking at this entire matter from a computer science perspective. We are not deeply familiar with the accounting literature on rounding. However, it appears that there simply are no well-agreed guidelines or best practices for how accountants should round numbers in real-life financial reports. If all the numbers are well-rounded (that is, they have less than one unit of rounding error in each reported number), the differences between different rounding algorithms will not be "material." Paying an accountant to think about such things for a few minutes costs more than a \$1 difference between rounding up and rounding down; and most official guidance seems to be focused primarily on requiring that the authors of reports *disclose* whatever procedure they used, rather than making any very detailed specifications of which procedure ought to be used.

Extreme cases, such as those described in our introduction, are statistically unlikely to ever occur; and if any statistically unusual pattern did appear in a report, it would be an audit issue anyway, warranting extra examination to determine why it had occurred even if the final answer turned out to be simple random chance. When rounded numbers on a financial report do not add up to rounded totals, the most common practice in real life seems to be to just add a note saying "Totals may not add due to rounding." and leave it at that.

Another alternative used in practice seems to be the application of an adjustment manually, somewhere, to make the balances add up; the accepted best practice seems to be to apply the adjustment to whichever number in the report has greatest absolute value, so that the relative error thus introduced (but actually, an error *removed*. . .) will be minimized, and the whole thing is excused by "materiality" again. It is implicitly assumed that one small adjustment will be enough. But this kind of ad hoc practice is obviously unsatisfactory for a computerized implementation without much more detailed specifications of when and how it can or should be done.

Discussions of rounding in a computerized context are often limited to technicalities of specific software: how to set up a given package to use a desired rule or precision level, asking what rule is the default in a given package, and so on. They also tend to be limited to first-order issues, such as whether an exact half-integer value should go up or down, considered by itself. Rounding in other fields, such as reporting results in experimental sciences, often aims to serve other goals, such as preventing the rounding from changing the overall average too much. Astrological rounding attempts to preserve distinctions in exact values that are relevant to interpretation and could be obscured by incautious application of rules used in other fields [5]. As we have described, no first-order rule is capable of making totals necessarily add up correctly.

However, making totals add up is a desirable goal, both because it makes reports more appealing and because it can prevent rounding from hiding other discrepancies that a report could reveal. In a double-entry ledger, the sum of debits and the sum of credits ought to be *exactly* equal, not only materially equal. If the debits and credits are reported as differing by one rounding unit, did that happen because of rounding, or for some other reason? It is worthwhile to know.

Computer scientists are also interested in worst cases even if they are unrealistic, and in exploring the limits of the capabilities of algorithms. We may also be sensitive to the legendary salami scam: someone modifies a financial system to tell the parties on each side of a rounded transaction that the rounding was in the other party's favour, and then deposit the difference into the scammer's account [2]. Dismissing rounding issues as immaterial leaves a firm vulnerable to such shenanigans.

For all these reasons, rounding seems worth thinking about. There is also an appealing symmetry: in computer science we often do rounding in order to approximately solve an $\mathcal{NP}$-hard problem [3]. Here, in the case of dags, we must solve an $\mathcal{NP}$-hard problem in order to do rounding.

To summarize, we have described the problem of rounding numbers to keep the rounding error small but also have the totals add up correctly. For totals structured as forests, we have shown that it is always possible to achieve correct totals with maximum rounding error less than one unit; that no better constant limit on maximum rounding error is possible; and we have given a $O(n \log n)$ algorithm for minimizing the maximum rounding error. We have also given a $O(n)$ algorithm that guarantees at least half the numbers in the forest case will be rounded to nearest and the rest to less than one unit, and an heuristic for that algorithm intended to increase further the proportion of numbers that are rounded to nearest. For totals structured not as a forest but a dag, we have shown that it is not always possible to have totals add up with maximum rounding error less than one unit, and we have shown that deciding this feasibility question is an $\mathcal{NP}$-complete problem.

Future work might go deeper into proving how many numbers can be rounded to nearest in the forest case; or examining whether there are useful limited classes of dags beyond forests on which the problem can be guaranteed to have a solution and solved quickly.

6 About this document

This is a draft intended for posting in the author's Web log. It is not clear that pursuing this as serious academic work is worthwhile. The draft is not formally peer reviewed and may be updated at any time; bear that in mind if you decide to cite it. News will be announced at http://ansuz.sooke.bc.ca/entry/334 .

References

[1] William J. Cook and André Rohe. Computing minimum-weight perfect matchings. *INFORMS Journal on Computing*, 11(2):138–148, 1999.

[2] Mike Judge (director). Office space, 1999. Motion picture.

[3] Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[4] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC 1978), May 1–3, 1978, San Diego, U.S.A.*, pages 216–226, 1978.

[5] Matthew Skala. Astrological charts with `horoscop` and `starfont`. *TUGboat*, 37(2):182–182, 2016.

[6] Theodore Sturgeon. The [Widget], the [Wadget], and Boff. *The Magazine of Fantasy and Science Fiction*, November–December 1955.