1 Problem Statement

Each year, thousands of high school students apply to undergraduate institutions for admission. Undergraduates apply to graduate schools. New Ph.D's apply for teaching positions. Those who succeed apply for government grants. Those who don't succeed apply for industrial jobs. In each case, a large number of submissions must be judged by a small number of judges. It is our task to determine a selection scheme that will guarantee that those submissions that get chosen are among the best.

There are too many submissions for each judge to read each one, so the papers are divided up so that each judge only has a small number to read (substantially less than the number of papers). After each paper has been evaluated, the judges' results are somehow combined, and the judges do another round of readings. This is repeated until the winners are determined.

Of course, the total number of readings should be kept small, and the work should be divided evenly among the judges. Also, the number of reading rounds should be kept small, and the total elapsed time short.

When we say "among the best" we mean this: there is an ordering of the papers to which the judges would agree, having read all of the papers. There is a definite answer to the question "Is this paper one of the best n papers?"

So, the problem is: Given some number of papers, P, and some number of judges, J, we must devise a scheme to select some number of winning papers, W. These papers must be from among the best $2W$ papers submitted.

For a competition such as the Mathematical Contest in Modeling (MCM), typical parameters would be $P = 100$ papers, $J = 8$ judges, and $W = 3$ winning papers.

2 Assumptions

1. The number of judges, J, divides the number of papers, P. If this is not initially the case, add dummy papers until J divides P. Dummy papers rank worse than all real papers.

2. Having read a set of papers, a judge is able to put them into an order corresponding to the "best" order, as mentioned above. This ordering is unique.

3. In the problem statement, it says "each judge will read some number of papers and give them scores." We assume that we may assign judges unequal numbers of papers, and some judges may be left idle.

4. Papers are indivisible. There is no way to read a paper in less time by using more than one judge.

5. The amount of time a judge spends on a batch of papers will be roughly equal to the time another judge spends on a comparably sized batch. Thus

if we distribute equal numbers of papers to the judges in the beginning of a round, we can expect them scored and returned at the same time.

6. A photocopier is available. This means that it is possible for any number of judges to read the same paper at the same time.

7. Any process other than reading and scoring takes negligible time and cost. This includes sorting based on scores, photocopying, and redistributing papers.

8. If a judge has read a certain paper, it need not be read again by that judge. That is, judges have perfect memories of the papers they have read. These memories can be used when sorting papers.

The quantity P/J appears often in this paper. The first assumption allows us to write P/J instead of $\lceil P/J \rceil$.

Although we may give judges uneven workloads and perhaps even leave some idle, we will make it a goal to even the workload as much as is reasonable.

The "memory" assumption can be implemented as follows. After reading a paper, a judge assigns it a numerical score. These scores are self-consistent: the better of two papers always gets the better score from the same judge. However, scores are in no way comparable between judges. They only help each individual judge to assign orderings.

3 Analysis of the Problem

A clear lower bound on the total number of readings is P. That is, one judge could read all of the papers, and put them in the best ordering. However, this does not match the goals of distributing the workload.

We will take care of a few easy cases right away. If there is only one judge available, then that judge reads and sorts all of the papers. The top W are then chosen. There is no opportunity to distribute the workload, and the judge must read a substantial portion (100%) of the papers. If two judges are available, they each read and sort half of the papers. They then exchange their favorite W papers, and read their new papers. The new papers are then sorted into the old list, and the top W are taken. Actually, only one judge needs to perform the second screening round. If W exceeds $P/2$, we simply pick W of the P papers at random; no judging is necessary. If $J = 3$, each judge reads and sorts $P/3$ papers, then hands the best W to the next judge, who reads and sorts them. The best paper overall will be the favorite paper of two of the three judges. The second best either occurs twice on the second row, or once on the first and once on the second. This identifies it uniquely. This sequence can continue until the best W papers have been identified.

4 Design of the Models

We will present two models. The first tries to adhere to the principle of having each judge read the same number of papers in a certain round, while the second model is not so worried about that idea.

4.1 The Shear-Shuffle Algorithm

The general idea of the shear-shuffle model of the selection scheme is this: we will establish bounds on where the best W papers can appear at each step, and use those to narrow our scope each time. Thus, the selection scheme depends heavily on which screening round we are currently processing. Also, we will use a very specific algorithm for redistributing the papers between each screening round.

To even the workload, we want to have all of the judges read the exact same number of papers during as much of the judging session as is reasonable. So, we will have a few massive screening rounds (three, it turns out) in which each judge reads the same number of papers. Then, we will switch to an "end-game" where each round will have some idle judges.

We will assume for now that $W < J - N$, where $N \approx \sqrt{2W}$. The need for this restriction, as well as the significance of N, will be described later.

In order to be fair, the algorithm must ensure that each paper is read. Since we want to distribute the workload evenly, it makes sense to have an initial round in which each judge reads P/J papers. So, our first step will be to distribute the P papers in piles of size P/J to the judges (Figure 1). We will represent these piles as columns. Each judge reads, scores, and sorts one column. The best papers rise to the top. We want to skim off some number of rows, and leave the rest behind. Now, consider where the W best papers could be. In the best case, each judge might have received only one of those papers, so the best W papers are all in the very top row. On the other hand, perhaps one judge received all of the best W papers. After sorting, they occupy the top W rows of that column, in which case that column should not be cut off above the W'th row. Since we cannot tell without further processing when either one of these cases has occurred, we must take the safe route and skim the top W rows (Figure 2).

What happens if there are not W rows to skim? This happens if $P/J < W$. This does not affect things; the columns are still sorted, and that is what we need. It decreases the actual number of readings that happen in the next round, but that is entirely acceptable. We will continue to do the accounting for the worst-case scenario, in which $P/J \geq W$.

Note that after the sorting, we are assured that the best paper has risen to the top row. The second best paper is either in the top row or it is in the second row, immediately below the best paper. This will be useful later.

At this point, each judge has read P/J papers, and the total number of readings is P. One screening round has passed, taking time P/J.

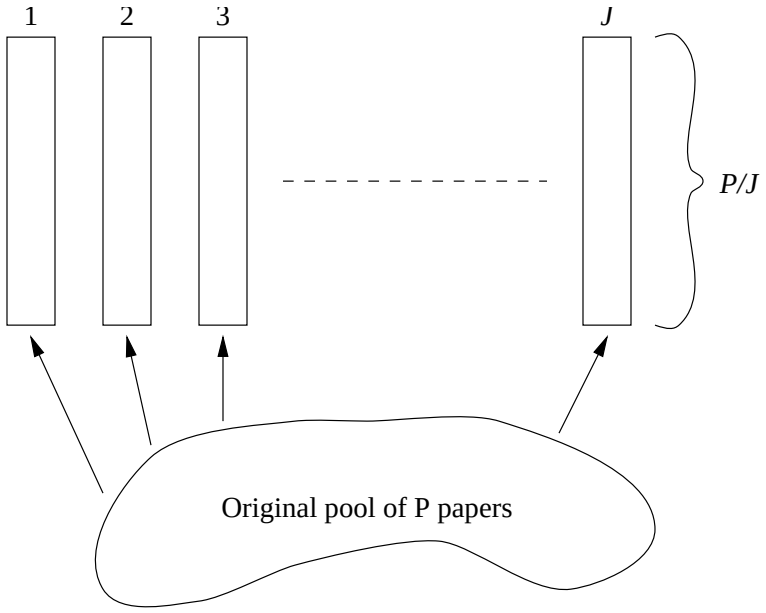


Figure 1: The initial distribution of papers

When we consider redistributing the papers, we are guided by a few principles:

- No judge should receive the same paper twice.
- No two papers should be compared to each other more than once. Such a comparison gives no new information.
- We want to be able to track the papers easily.

Taken together, these three principles eliminate using a totally random redistribution scheme, because it would certainly violate the third idea, and might violate the other two.

Now, the papers are redistributed according to a rule that amounts to a horizontal shear. The top row of papers moves one column to the left, and the leftmost paper wraps around to the rightmost column. The second row of papers moves two columns to the left, with wraparound, and so on: the W'th row (the last one we are still considering) moves W columns to the left. If W is too large, papers will wrap all the way around and back into the columns in which they started (Figure 3). Thus, for the first round, we restrict $W < J$. The second round will cause a further restriction, as mentioned above and explained below. The MCM parameters meet both restrictions.

Once the shuffling is done, the judges read, score, and sort their new columns of papers. At this point, each judge has read $P/J + W$ papers, and the total

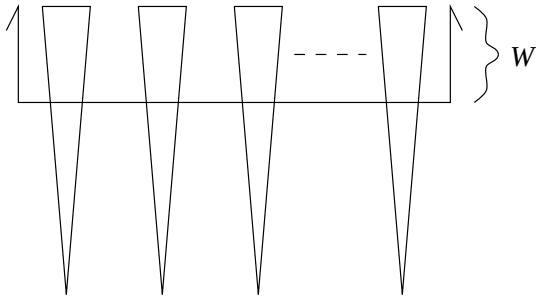


Figure 2: Having sorted the papers, the top W are used

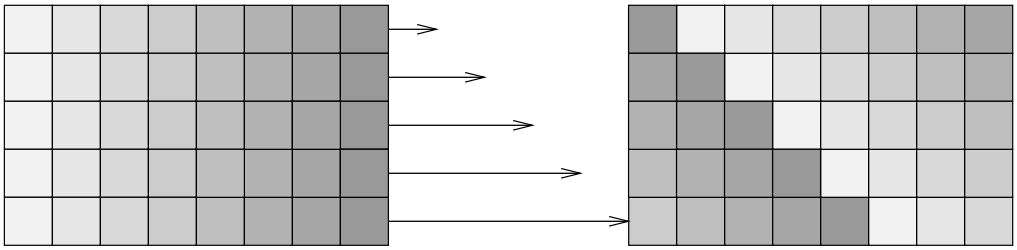


Figure 3: The shear-shuffle for redistributing papers

number of readings is $P + WJ$. Two screening rounds have passed, taking total time $P/J + W$.

Consider what happens to the two special initial distributions mentioned above. If the best W papers were distributed one per judge, they moved to the top row in the first round, stayed there during the shuffle (but moved to the left), and stayed in the top row during the next sorting. If the best W papers were all in one column, they stayed there during the first sort, but then were each given to successive judges during the shuffling. The sort then moved each of them to the top row. One might hope that all of the best papers are in the top row now, but this is not the case.

It is possible to construct an initial distribution which will result in at least one of the W best papers being placed in the row $N = \lfloor \frac{1}{2}(\sqrt{1 + 8W} - 1) \rfloor$. It is not possible for one of the best W papers to appear any lower than row N. These results are derived in the appendix. For the MCM parameters, $N = 2$. Since we do not want to eliminate any of the best W papers, we must skim row N and above for the next round. We shear-shuffle these N rows, then have the judges read, score, and sort their new columns of papers. At this point, each judge has read $P/J + W + N$ papers, and the total number of readings is $P + WJ + NJ$. Three screening rounds have passed, taking time $P/J + W + N$.

Now we shift strategies and do a lot of information processing with no readings. Recall that, by assumption, information processing takes negligible time and cost. We recall all of the papers that judge j has judged, and sort them

according to the scores that judge j gave them. We put these into column j, with the best paper at the top. Doing this for each j gives a sorted table with the following nice properties:

1. No paper appears twice in the same column.
2. Each of the best W papers are in the table exactly three times.
3. The best paper occurs only in the top row, exactly three times.
4. None of the best W papers appears below row W.

The restriction $W < J - N$ yields property 1. In a worst case scenario, a paper can only start in row W and move W columns on the first shear-shuffle. Since only N columns are shuffled in the second round, the farthest a paper can travel across columns is $W + N$. By the restriction $W + N < J$, a paper can not wrap around and be read by the same judge twice.

Property 2 is valid since each of the best W papers passed each of the two skimming operations, and was also in the initial data set (a total of three occurences, which must be in separate columns by property 1). Since each column is sorted and properties 1 and 2 hold, properties 3 and 4 are immediate corollaries.

Unfortunately, the table can also have the following undesirable properties:

1. A paper that is not among the top $2W$ can occur three times above row W. Indeed, it can even occur in the top row three times.
2. A paper that is between W and $2W$ can occur below row W.

Given these properties, we want to select W papers from the top W rows, but we must first eliminate those papers that are worse than $2W$. We will attempt to do this in two stages. First, we cross out every occurrence of any paper that appears below row W, since it is definitely not one of the best W papers. This might leave some gaps in the table, where paper y does not appear below row W, but paper x does, and paper x appears above paper y in another column. In this case, since x goes, so must y. Then, we cross out all other occurrences of y, and look for gaps below them. We continue until we can't cross out any more papers. We will never be able to cross out the best W papers, and we started with a finite number of papers, so this method will eventually terminate.

With a bit more effort, we can eliminate even more papers. If a paper has W or more distinct papers appear above it in the table, it cannot be one of the best W papers. So, for each remaining paper, we make a list of the papers which appear above it. Now, we might notice that paper x appears above paper y, and paper y appears above paper z, but in a different column. In cases like this, we add x to z's list. Once each list is complete, we eliminate each paper that has a list longer than W.

These two will usually eliminate most of the papers that are worse than $2W$. However, it is not guaranteed to eliminate all of them. Indeed, if a paper ends

up in the top row three times, it will not be crossed out, and its list will be empty. It is temporarily indistinguishable from the best paper.

Now that we have used all of the information from the three screening rounds, we must go back to the judges to get more information. We don't want to continue the shear-shuffle-sort procedure, because it requires too many readings. In addition, it is very hard to guarantee that the W papers selected are from among the top $2W$. Instead, we will shift to a different method.

Notice that any paper that appears three times in the top row could be the best paper; we will compare them to each other and find out which is the best. Then, we will find the second-best paper, and so forth. We will show that this is not as expensive as it might seem.

Consider all of the papers that occur in the top row three times; these are all contenders for being the best paper. There are at most $J/3$ of them, and each one has been read by 3 judges. So, to determine if paper A is better than paper B, we make a copy of B and give it to one of the judges who has already read A. We can similarly give copies of papers C and D to the other two judges who read A. We also make sure B and C are directly compared, as well as comparing B to D and C to D. The pairing is shown in Figure 4. This determines the ordering of the four papers completely. All of the readings can be done simultaneously, and each judge does at most one reading.

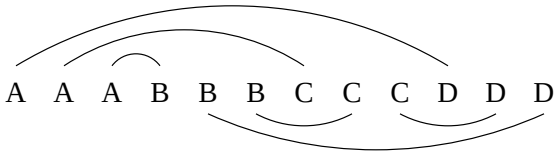


Figure 4: How to sort 4 papers

If there are more than 4 contenders, we split them up into groups of 4 and run these comparisons simultaneously. We take the best from each group, then put those into groups of four, compare them, and so forth (Figure 5). Each box represents a contender, which consists of three copies of one paper. This is a tree where each node has at most 4 branches, so it has height at most $\lfloor \log_4(J/3) \rfloor$, in the worst case. This gives us the best paper. Notice that in the figure, we have 24 initial candidates for the top position; this means that there are at least 72 judges. Even with this large number (relative to the MCM), it only takes three levels to determine which paper is the best, and there is room to add $8 + 16 + 16 = 40$ more candidates (120 more judges!) before another level is necessary. So, this tree-based procedure is very efficient.

Now that we have the best paper, we bump all of the other contenders down one row. This means that the second-best paper occurs at least three times in the second row, along with other contenders. We run the same 4-based sorting

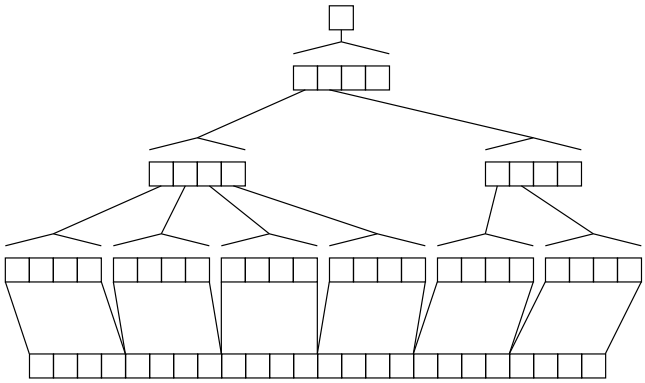


Figure 5: Sorting $J/3$ papers in groups of 4

algorithm, and determine the second-best paper. We bump all of the other contenders down one row, and repeat until we have determined all of the best W papers. Along the way, we might find that we have eliminated enough other papers to be able to stop the end-game and just pick the remaining papers.

In the absolute worst case, we failed to eliminate any papers with the crossing-out and listing methods. We also ended up with $J/3$ contenders in the top row, and in every row after that. We have gone through W of the 4-based trees, each of which has $\lfloor \log_4(J/3) \rfloor$ levels, and each level happens as quickly as one judge can read one paper. So, we have an extra $W \log_4(J/3)$ reading rounds. At the first level of each tree, each judge reads a paper. At the second level, only $J/4$ judges read papers. At the third level, $J/16$ read papers, and so on until only one paper is left. We will bound the sum of the readings with the sum of the infinite series $J(1 + 1/4 + 1/16 + \cdots) = J/(1 - 1/4) = 4J/3$. This happens each of the W times we use the 4-based tree, for a total of $4JW/3$ extra readings in the end-game. Per judge, this is $4W/3$ readings in the end-game.

When we have finished the shear-shuffle algorithm, we have the following worst-case data:

Method	Reading Rounds	Readings per Judge	Time Elapsed
Shear	$3 + W\lfloor \log_4(J/3) \rfloor$	$\frac{P}{J} + \frac{7}{3}W + N$	$\frac{P}{J} + \frac{7}{3}W + N$
Shear (MCM)	6	22	22

Notice that the number of readings per judge is equal to the time elapsed, since we have made the worst-case assumption that all of the judges are occupied during each round of the the end-game, and each round takes a constant amount of time.

What can we do if $W \geq J - N$? One possibility is to split the problem into two subproblems: first, we find the best $W/2$ papers. Then, we remove them from the pool and find the next best $W/2$ papers. Of course, if $W/2 \geq J - N$, we split it again, and so forth. Although the splitting process seems to be inefficient, it is not much worse than the model without the split. The first screening

round is the same as the non-split model, and so it takes P/J time. The second iteration only includes $W/2$ papers from each column, instead of W. To analyze the third screening, recall the approximation $N(W) \approx \sqrt{2W}$. In this case, we are only using half as many papers, so we have $N(W/2) \approx \sqrt{W}$. The end-game is also half the length of the non-split model, so it takes time $\frac{W}{2}\lfloor \log_4(J/3) \rfloor$. This gives a total elapsed time of approximately $P/J + W/2 + \sqrt{W} + (W/2)\lfloor \log_4(J/3) \rfloor$ for the first half of the problem.

Now we have found the best $W/2$ papers. We eliminate them from the initial distribution we had before, and move papers up to fill the gaps. This requires no readings, so no extra time is expended. If a column moves up so much that it is less than $W/2$ in height, we add dummy papers to the bottom. We now know that the overall second-best $W/2$ papers are among the top $W/2$ rows, so the analysis can proceed as before. Thus, we have a further cost of $W/2 + \sqrt{W} + (W/2)\lfloor \log_4(J/3) \rfloor$, for a total elapsed time of $P/J + W + 2\sqrt{W} + W\lfloor \log_4(J/3) \rfloor$, compared to the previous time of $P/J + W + \sqrt{2W} + W\lfloor \log_4(J/3) \rfloor$. The difference is approximately $(\sqrt{2} - 1)N \approx 0.4N$, which is probably insignificant in relation to $P/J + W + W\lfloor \log_4(J/3) \rfloor$. This will happen again if we must split the problem into four subproblems, each of size $W/4$. However, it will be a rare contest that has that many winners in relation to the number of judges.

The shear-shuffle algorithm is rather complex, and it is not easy to visualize the various data paths and paper shuffling it requires. To improve the situation, we offer a simpler algorithm that does not generally do as well at keeping the judges occupied.

4.2 The Unbalanced Tree Algorithm

If we are not so concerned with using each judge at each step, we can construct a simpler algorithm to find the best W papers. We shall show that the new scheme needs fewer total readings than the shear-shuffle algorithm, and distributes the total readings so that every judge does essentially the same amount of work. However, the total elapsed time will be greater, and it will end with one judge doing the final scoring while the rest are idle.

The crucial structure for the implementation of this new scheme is a tree, with some number of judges in clumps at the bottom reading a specified number of papers and passing up W to the next tier, consisting of a smaller number of different judges. This process continues until a final batch is passed up to a single judge who can simply pick out the best W papers as winners.

The total number of readings in such a tree can be calculated as follows. Let R_j be the number of papers read by judge j. Since each judge passes up only W papers, judge j eliminates $R_j - W$ papers. At the end, there are W winners, so $P - W$ papers were eliminated. Thus, we have

$$P - W = \sum_{j=1}^{J} (R_j - W)$$

which, when solved for $\sum R_j$, gives

$$\sum_{j=1}^{J} R_j = P + (J-1)W$$

Thus, the total number of papers read is $P + (J-1)W$.

We shall build an initial tree schedule for grading papers in the following manner. Refer to Figure 6 as the algorithm is developed. Start with the topmost judge j_0 and number of papers, L_0 to be just greater than or equal to the average number of papers, so

$$L_0 = \lceil (P + (J-1)W)/J \rceil = \lceil P/J + W - W/J \rceil$$

Find the greatest multiple of W (call it δW) such that $\delta W \leq L_0$. This δ will be the maximum number of judges we want to have passing papers to j_0. If $\delta W < L_0$ record the quantity $L_0 - \delta W$ and set this many papers aside from the initial stack of papers in a "reserve bin". These papers will remain unread until j_0 picks them up. Thus j_0 reads L_0 and we are still as close as possible to perfectly even distribution.

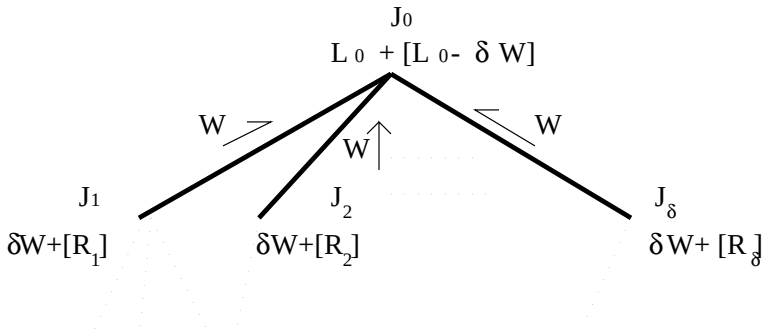


Figure 6: Constructing a judging tree

Proceeding down from j_0, we start by drawing δ edges down to δ more judges. It is possible during this step for the total number of judges included in the tree to exceed the total number of judges available, J. In this case, we just stop drawing edges as soon as we reach the maximum. If we reach J judges, every judge with no underlings is initially assigned $\lfloor (P + (J-1)W)/J \rfloor$ papers to read from the initial pool. Falling short of δ edges means, of course, that j_0 isn't receiving a full share of δW papers from the lower level. This is easily compensated for by increasing the number of papers in j_0's reserve bin to make up the difference. Finally, start adding papers, one per judge, to the lowest level first so that the overall total number of papers read is the correct total. If the current total of papers is too high, start removing any extras lying around in reserve bins. As a last resort, remove any extra papers from the bottom layer. When the total number of papers read is P, the construction is complete.

If we do not yet have J judges included in the tree, treat each judge in this second tier as we treated j_0. At each judge, we attempt to draw δ edges, completing each judge's set before moving on the next judge. When we have assigned J judges, every judge in the entire tree with no underlings is assigned the same number of papers. The structure of the tree is complete. Since we constructed it breadth-first, the height is $H = 1 + \lceil \log_\delta J \rceil$.

Now we finalize the distribution of papers among the judges. Judges with underlings each receive at most δW papers from lower levels, and have their own reserve bins adjusted so that their paper totals are up to L_0. Judges with the same number of levels below them in the tree are assigned to the same round, so all judges with no underlings read their papers in the first round, all with one level of underlings read in the second round, and so on. Starting with the lowest layer, extras are again added in, one per judge, to bring the total papers up to P.

We argue that this scheme is optimal in the sense that it divides readings as evenly as possible among all the judges and, subject to this even division, gives the tree of minimal height. The "even-division rule" is followed since each middle judge starts with above the required average number of papers, each bottom judge starts with below this average, and additional papers are added and subtracted in appropriate places to make the total correct. It gives the tree of minimum height since it starts at the top and, at each step, builds breadth first with the maximum possible number of edges (subject to the even-division rule).

4.3 The Balanced Tree Algorithm

The unbalanced tree is far from optimal in terms of the total time required to complete the judging. Each tier must wait for all of the tiers underneath it to finish their processing. A few of the judges have reserve bins that they can read from while they are waiting, which will save time when it becomes their turn. We will try to make use of these bins, while keeping the same height and the equal division rule. This tree, however, will try to minimize the number of papers that must be passed up from the bottom. The rest of the papers must be put into the bins, where they will ultimately save time.

Start by constructing a simple binary tree of height H. If this includes J judges, we're done. To include fewer judges, start taking away judges at the lowest level, moving judges down if they no longer have any underlings. To include more judges (the more likely case) start at the second tier up from the bottom and add single edges to the judges in this second tier. This will make three for each. If we still need to include more judges after that pass, move up a tier and add a single edge to one of the above judges, with up to three edges below that edge (to make that judge's underling total equal to the other judges in the same level). Continuing in this manner, we exhaust one level before we move on to the next. If we reach the top of the tree without having J judges, start over at the bottom of the tree by adding a fourth edge to every second tier judge. We can continue to do this until every judge not in the bottom row has δ

underlings, a tree that has a least as many nodes as the tree constructed above (which had height H, and each node had $\leq \delta$ underlings). Of course we stop if, at any time, we have J judges. Overshoots are possible when new branches are added, and are compensated for by simply removing the correct number of edges from the new branch. The structure of the tree is now complete.

Constructing a tree in the above manner assures us that we will obtain a tree which is as close to balanced as possible. Any additional edges beyond a perfect k-ary tree are pushed as low as possible in the tree, where $k \leq \delta$ is as low as possible. We distribute the papers by giving the bottom level judges $\lfloor (P + (J-1)W)/J \rfloor$, and throwing the rest into the reserve bins. This graph minimizes the number of papers getting passed up to each judge, with the stipulation that no judge is passed significantly more papers than another in the same level. It is here we have the opportunity for upper level judges to gain information before new papers are passed to them, since nothing is stopping them from reading papers in their reserve bins before anything is passed to them. Thus the first round would consist of the bottom level judges reading their entire initial batch of papers, while the upper level judges read the papers in their reserve bins. As the papers are advanced up the tree, the papers passed to each judge are read and sorted in terms of the papers that judge has already seen, and the time each judge requires is only the time required to read the newly passed papers.

Consider the tree algorithm, in both its incarnations, operating on the problem specified in the original problem statement, namely $P = 100$, $J = 8$, and $W = 3$. Constructing the tree using the original method, we see that $L_0 = 16$, so that $\delta = 5$, since $15 = 3 \times 5$. In Figure 7 this works out, in its conclusion, to have five branches going to the top judge, with four of those simply going to the lowest levels. The numbers of papers that each judge has to read at each level are 15, 6, and 15. This yields a elapsed time $15 + 6 + 15 = 36$ paper-reading time units.

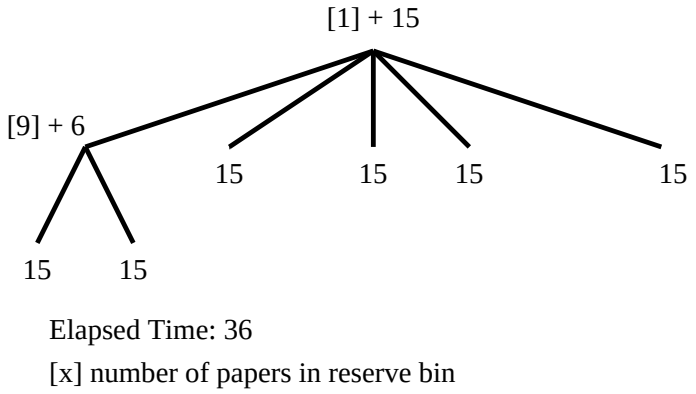


Figure 7: An unbalanced tree, constructed with MCM parameters

Attempting to optimize the elapsed time for the same problem results in

Figure 8. The only addition to the original binary tree skeleton is the dotted line adding a new judge at the bottom. Again counting the number of readings that have to be conducted in each round, we count 15, 9, and 6. This gives a total of 30 time units for this particular grading schedule. Note that the best time we get with the tree is still significantly greater than the absolute upper bound for the shear-shuffle algorithm applied to the same problem.

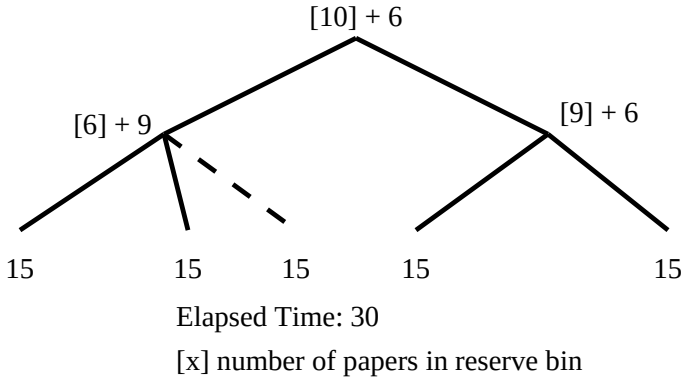


Figure 8: A balanced tree, constructed with MCM parameters

Here are the upper bounds and data from the MCM:

Method	Reading Rounds	Readings per Judge	Time Elapsed
Shear-Shuffle	$3 + W\lfloor \log_4(J/3) \rfloor$	$\frac{P}{J} + \frac{7}{3}W + N$	$\frac{P}{J} + \frac{7}{3}W + N$
(MCM data)	6	22	22
Unbalanced tree	$1 + \lceil \log_\delta J \rceil$	$P/J + W - W/J$	$L_0(1 + \lceil \log_\delta J \rceil)$
(MCM data)	3	15.125	36
Balanced tree	$1 + \lceil \log_\delta J \rceil$	$P/J + W - W/J$	$L_0(1 + \lceil \log_\delta J \rceil)$
(MCM data)	3	15.125	30

5 Model Verification

Both algorithms work well for the parameters common for the MCM. They were developed using P, J, and W as variable parameters, instead of being fixed at the particular values that are common in the MCM judging, so both algorithms scale well when the parameters are increased somewhat proportionally. At the outset, we gave samples of what to do for extremely large values of W, and what to do when very few judges are available. The shear-shuffle algorithm works most efficiently when $W < J - N$, but it still works well when this condition is not met. The tree algorithm has no such limitations.

5.1 Statistical Properties of the Shear-Shuffle Algorithm

For many cases of the shear-shuffle, there will be no need to use the end-game based on the 4-sort. Using 15,000 data sets (random parameters similar to those of the MCM, with random initial paper distributions), we found that the number of remaining papers, P', after a shear-shuffle is relatively small. In fact, the linear regression line is

$$P' = (1.08)W + 0.94$$

with the correlation coefficient $\rho = .89$. This means we will be likely to have $W + 1$ papers remaining after the shear-shuffle, which we could ask one judge to read and sort, instead of performing the 4-based tree sorting. This means that in almost all practical cases, the upper bound on the number of readings per judge is $P/J + W + N$, with an extra $W + 1$ for the final judge.

5.2 Sensitivity Analysis

So far, we have considered the judges to be absolutely accurate in their judgments. If the possibility exists that they occasionally transpose two adjacent papers, we can adjust our algorithms to compensate. More specifically, suppose that each judge might always judge two adjacent papers to be in the wrong order, and that this can only occur to one set of papers per judge. If the transposition occurs in the bottom row, we need to skim $W + 1$ at the first screening and $N + 1$ at the second screening. This increases the size of the end-game a little bit, but not very much. Now that we are playing it safe and skimming extra papers, suppose that the transposition actually occurs at the very top: one judge switches the best two papers. This means that the best paper will not appear three times in the top row, so it will not appear to be a contender. It is possible that the second-best paper will appear to be the best, in which case it pushes the best paper down to the second slot. It then gets selected, so we still get the best W papers, only in a slightly different ordering. However, the order of the final papers does not matter. Anywhere along the way to the final rounds, we might end up with a contradiction, such as a paper being better than itself, or a non-transitive relationship. In these cases, the judging process must stop and extra judges must read the papers in question to fix the problem.

The tree algorithms are not as sensitive to transpositions. They must use $W + 1$ instead of W for the same reason that the shear-shuffle algorithm must, but they are not sensitive to transpositions at the top.

6 Strengths and Weaknesses

- The shear-shuffle algorithm is rather complex. The tree algorithms are much simpler, and have an easily visualized flow of papers.
- The tree algorithms do not keep all of the judges occupied at each round. Neither does the shear-shuffle algorithm, but it comes much closer.

- All of the algorithms guarantee that the best W papers will be chosen.
- All of the algorithms give upper bounds on the time it will take to finish, once the trees are constructed. The tree algorithms always meet that bound exactly, while the shear-shuffle algorithm can quite often do better, depending on the initial distribution.

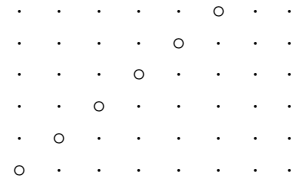
6.1 Recommendations

We recommend the balanced tree algorithm. This approach has a deterministic amount of elapsed time, in addition to maintaining the fewest readings per judge. Certainly, the balanced tree algorithm is superior to the unbalanced tree algorithm, since all positive characteristics of the unbalanced tree are also inherent within the balanced tree, and the balanced tree takes less time. Since the shear-shuffle algorithm appears to take less time in the example case ($J = 8, P = 100, W = 3$), this would be ideal for a judging process that must happen quickly. However, if time is not a significant constraint, and if judges don't mind sitting idle, the balanced tree algorithm is superior to the other two.

7 Appendix: A derivation of N

We will attempt to construct a worst-case initial distribution of the papers going in to the second round of the shear-shuffle-sort algorithm. By worst-case, we mean a distribution which requires that we skim off the maximum number of rows. We are worried about saving each of the best W papers, so we are interested in how far down in the table any one of those papers is. In the first round, we saw that this was a stack of the best W papers, all in one column. We will represent the best papers with circles, and otherwise unremarkable papers with dots.

Where could such a distribution come from? We apply the reverse of the shear-shuffle to find



This is the table that was produced by the sorting, which means that any paper must have better papers above it, to "push it down" in the table. However, we

have already used up all W of the best papers to make the diagonal. So, the sorting would have collapsed them all up to the top row. Clearly, we cannot have the best W papers all in the same column during the second round.

What is the maximum depth of the best W papers? We must start filling in above the diagonal to make a triangle. Since we have only W papers to work with, we can compute the size of the tallest triangle possible. Notice that we are dealing with triangular numbers, which have the formula $n(n+1)/2$ where n is the height. We find the maximum integer n such that $n(n+1)/2 \leq W$. So we write the quadratic equation

$$\frac{1}{2}n^2 + \frac{1}{2}n - W = 0$$

and solve to get

$$n = -\frac{1}{2} \pm \sqrt{\frac{1}{4} + 4 \cdot W \cdot \frac{1}{2}}$$

which simplifies to

$$n = -\frac{1}{2} \pm \frac{1}{2}\sqrt{1 + 8W}$$

Since $\sqrt{1+8W} > 1$, we have one positive and one negative root. We take the positive root. Recall that the height must be an integer, and we want the greatest n such that $n(n+1)/2 \leq W$. If we round up, we might exceed W, so we round down. Our final formula is

$$N = \lfloor \frac{1}{2}(\sqrt{1+8W} - 1) \rfloor$$

Note that for large W, this grows with $\sqrt{2W}$.