strategy-FICO

1 message

Tianxiang Liu <ivan.liuyanfeng@gmail.com> Thu, Dec 11, 2014 at 9:26 AM
To: Tianxiang Liu <ivan.liuyanfeng@gmail.com>

Begin by noting that in the worst case that it will take 14 ~ log (16) / (10 * log 1.02) days to re-train from the lowest skill level of 0.25 to to the highest 4, by working consistent 10 hour days. As well, initially it will take only 7 ~ log (4) / (10 * log 1.02) days to train from the initial skill level of 1 to the highest skill level of 4. Furthermore a single 12 ~ 10 - (10 * log 1.02) / log 0.9 hour shift results in no change in skill level, albeit requires a ~2 hours later start to the next shift. Finally all build shifts longer than 52 hours ~ -24 * log(16) / (10 *log(1.02) + 14 log(0.9)) deplete the skill maximally from 4 to 0.25; remembering that any build shift longer than 24 hours can be started earlier to maintain the 10:14 ratio between sanctioned and unsanctioned time.

We can conjecture that the optimal strategy involves the following two cycles:

1. Punch-Rest-Train. Where a long shift (the punch) is taken to build a multiday item, followed by the mandatory rest, and at most 14 days of re-training.
2. Diminishing shift extensions. This exploits the fact that there are multiple solutions to x* log 1.02 + y * log 0.9 = 0. For cycle 2 we can generate the following sequence to describe shift duration di of the i shift. Letting a = - log 1.02 / log 0.9, be the skill neutral shift extension factor we have

d0 = 10
d1 = d0 * (1 + a)
dn+1=d0 + a * (d0 - dn)

This is an asymptotic sequence, and finite under the truncation to the the nearest minute. The limiting duration is 600 minutes but with the magic offset of 1.6 ~ 10 * log(1.02) / log(1.02/0.9) hours, or 94 minutes. We can find a net gain in 6 iterations:

d0 = 600 minutes
d1 = 712
d2 = 578
d3 = 604
d4 = 599
d5 = 600

For a total of 3093 minutes worked in 5 days, a 93 minute gain over regular shifts, with no penalty. This also allows for penalty free work on 4 * 712 = 2848 minute items. Now to reset the shift would require the loss of 94 minutes (that seems suspiciously close, is there a lemma in there, or was it by design?). This means that to make the most of the offsetting strategy one needs to end the offsetting by taking on a large item that exhausts the skill. The exact number of days to go before ending each offsetting depends on the distribution of durations.

So we can conjecture that the skill can be maintained at 4, and thus we can break our items down by their build duration **d**:

d < 2.5 (full exhaustion safe)
2.5 <= d < 10 hours (incremental re-training)
10 <=d < 40 hours (can be built in a day, by maximal trained)
40 hours <= d < 48 hours (requires cycle 2)
48 hours <=d < 96 (requires cycle 1, only unsanctioned time increases)
96 hours <= d < 208 hours (requires cycle 1, sanctioned:unsanctioned constant)
208 hours <= d (requires cycle 1, re-train time is maximally 14 days)

Now as long as the system is sufficiently overloaded (sorry that is a bit of loose term), the assignment of the particular cycles, and the build blocks described above is largely arbitrary. From that you can combine the time estimates with the distribution of duration to generate a good lower bound on the best possible score for the optimal strategy. This also allows for a semi-exact relationship between the stopping time and the number of workers (especially in the large number limit). Furthermore the accessibility of the lower bound is controlled by the ratio of the type 1 & 2 durations, to the type 5 & 6 durations. Clearly if there is an insufficient number of type 1 & 2 durations the skill level can never be re-trained.

Overall this hints at the following **algorithm**:

1. Block items into 2.5 hour, 10 hour, 40 hour, 48 hour, and overwork days, where all the items have close to the same arrival time
2. Do not start work until all the items have arrived, using the proportions of item build durations create cycles of days ending in skill exhaustion, and nit these together for each worker
3. Reorder and swap between workers days of the same length so that the earliest arrivals are worked on first
4. Push the start work times back to the arrival times, while maintaining day ordering form step 3.
5. Decimate work force by stealing work to fill in the gaps created by pushing back in step 4.

The whole process is rate limited by the the arrivals between October 22 and November 6. It follows that the optimal solution time will be dominated by O(1/n), in the number of workers n, from which the extreme is found from ln(1+n)=n / (1 + n), where we have absorbed the base-10 logarithm into the rate constant. Which interestingly means the number of workers is not only independent of the work rate, but that 2 workers will result in the maximum score. It follows that 900 workers will always be advantageous. This is an idiosyncratic result due to the choice of logarithms in the scoring function, and was perhaps an unintentional result of the patch up of the scoring mechanism we saw earlier.

This also requires a quick analysis to see if build-rest-train is faster in the long run then just working at 0.25 < a < 4, and if so what is the optimum? Again not hard just requires pulling the appropriate statistics. The total length of an average exhaustion cycle in days is:

dtot = 2.4 * dmax / a + log (4 * a) / (10 * log 1.02)

where dmax is the average length of the maximum exhaustion items. Reading off the attached graph we have dmax = 14400 minutes = 10 days. This yields the limits of dtot = 96 for a = 0.25, and dtot = 20 for a = 4. Working through the derivative we find a = 4.75 = 240 * log 1.02 is the maximum, however this is disallowed, and so the optimal training is on the boundary a = 4.

Again this admits the **following algorithm:**

Count up the number of exhaustion tasks ne ensure 14 * ne worth of training days are either set aside from earlier, or will be available on demand.
For 900 workers cycle through the exhaustion tasks.
Assign the remaining tasks to the time before the onset of the exhaustion tasks.
The bolus of 1,082,880 maximal exhaustion items, with a total duration of 18,409,417,194 minutes, requires at least 25368 days = (24 * 18,409,417,194 / (40 * 1440) + 1,082,880 * 14) / 900, or 66 years.
This is nearly a 2.1 fold increase over the naive estimate for the lower bound of 12039 days.

Note that by integrating the effective minutes over the 14 training days we have 10264 minutes available. This allows for 11,114,6680,320 minutes = 1082880 * 10264 for building all other items. In comparison there is less than one half as many minutes left in all the other items 4,637,418,394 = 26,003,950,765 - 21,366,532,371.

In light of the under allocation of training items we can re-cast this as a calculus of variations problem by taking the limit to the continuum in item duration. Subject to the constraint of the distribution of durations, we can find a function from each duration to the fraction of the training time used such that the total time is minimized. To give an idea of what this means consider a **few corner cases:**

Only fully train the 1/2 longest exhaustion tasks
Train all exhaustion tasks only out to 1/2 of the available training time
Only train the 1/2 earliest exhaustion tasks (trivially ignored because of the lower limit of 66 years)
There is an equilibrium ability 'a' that can be found. We exploit the freedom to move the start time of an item to before a sanctioned time so that the ratio of sanctioned to unsanctioned time is always 10:14, further we note that any choice in 'a' restricts the available items used to train back to 'a', thus:

Training required to maintain 'a' = Training available to maintain 'a'

Now for any choice of 'a' the minimum full exhaustion time is da=-1440*log(4a)/(log(1.02)+log(0.9)), we can then divide the durations into **three half open intervals:**

[1, a * 600) dt = 0
[a * 600, da) dt = a * (1.02^(10 * di / 24 * a)) * (0.9^(14 * di / 24 * a)) * 60 exp(-di(10 log 1.02 + 14 log 0.9)/(a * 1440)-1)
[da, \inf) dt = 60 * (4a -1) / (4 * log 1.02)
Where dt is the required training time to restore back to 'a', after work di, and was found by working through the first order ODE of the generator equation for work accomplished versus actual time spent.

(* I apologize that the current ability graph is incorrect, I will be re-running the statistics shortly)

Unfortunately it seems reasonable to estimate that the optimal solution will exploit weakness in the distribution of the items duration and arrival, and the idiosyncratic choice in scoring, rather than finding an interesting heuristic for a long standing CS problem. This problem seems to be more of one of reverse engineering than actual linear programming or optimization.

Thank you for that analysis. I am working on a solution with different worker states: "Train", "Overwork", "Re-train" etc, switched depending on available orders to process. My initial thoughts are that it will be impossible to re-train 0.25s back to 4.0s with the available smaller toy orders, and it will be a matter of figuring out whether a productivity of 0.5 (for example) is a reasonable goal instead.