Solution of the team "Master Exploder"

Marek Cygan and Marcin Mucha

February 7, 2015

1 The problem

The problem to solve in this year's contest was a variant of machine scheduling. However, instead of machines performing jobs, we had elves producing toys for children, which makes much more sense. The objective was to minimize the product of the total makespan and the natural logarithm of the number of elves used (out of maximum of 900) plus 1. Once started, toy production cannot stop, i.e. no preemption is allowed. Each toy has an arrival time, i.e. the minimum time when it can be started, but there are no deadlines. Each toy also has a duration required to produce, expressed in minutes (we will also often use the term *size*, instead of *duration*).

The defining feature of the problem was the way elves' speed (called *rating* in this problem) worked. The time was divided into alternating intervals of: 10 hours of working-hours (starting at 9:00 each day), and 14 hours of off-hours (starting at 19:00). When elf produces a toy and spends a total of a working hours and b off-hours, it then has to rest for b working hours before starting the next toy. This seems almost equivalent to saying that elves only work during working hours (except that the very last toy can be produced during off-hours as well). However it is not. The trick is that the numbers a, b influence the evolution of the rating of the elf producing the toy. This rating remains constant while producing a single toy. However, after the toy is finished it is immediately updated using the following formula:

$$rating_{new} = rating_{old} \cdot 1.02^a \cdot 0.9^b.$$

If the new rating is not in the interval $[0.25, 4.0]$ it is rounded to the nearest point within this interval. All elves start with a rating of 1.0.

2 Data analysis and basic observations

What we did first is look at the data.

A basic estimate. The total size of all the toys is around $2.6e10$. Even assuming that all of them are produced at the rating of 4.0 by 900 elves, the production is going to take at least 33 years. This has at least two implications.

The objective function. Since the objective function penalizes us less and less for using each next elf in the solution, the only possible reason to not use all 900 could be that we can start producing the last toy at its arrival time (or close to it) with less than 900 elves. This is clearly not the case. Therefore, we should always use exactly 900 elves, and the objective function reduces to minimizing the total makespan.

The arrival times. As it turns out, our estimate of 33 years is extremely optimistic, the actual values are about 10 times larger. Still, even this estimate is significantly larger than the largest arrival time (all toys arrive during the first year). Since the arrival times complicate the problem quite a bit, it makes sense to completely ignore them and separately process the first year where they do matter.

More in-depth look. It is quite clear that the basic strategy should be to speed elves up using small toys. Speeding up from the rating of 0.25 to the rating of 4.0 takes quite many toys and a quick look at the distribution of toy sizes suggests that these small toys are going to be a bottleneck, i.e. we do not have enough of them to speed up the large toys as much as we would like. The small toys should therefore be treated as a resource, and indeed this is the way we were thinking about this problem.

More on arrival times. Fig. 1 shows the number of toys of of different sizes arriving in each week, whereas Fig. 2 shows proportions of total size of toys from the given category.

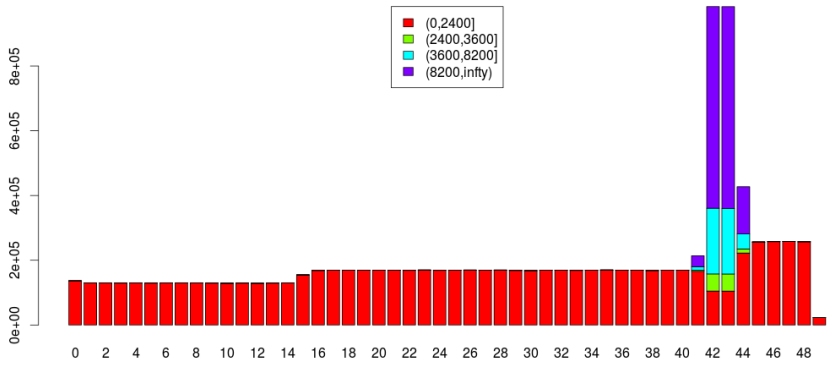

Figure 1: Number of toys in 4 different size ranges released in each week.

The thresholds for the groups in these figures has not been chosen arbitrarily, as will become clear later on. The shapes in these figures will also play role in the way we play out the first year. For now, let us just note that until week 41 almost nothing happens and then suddenly we get almost all toys at once. In a sense, this is good news, as this means that ignoring the arrival times is not going to hurt our score too much. This is because for most of the first year not

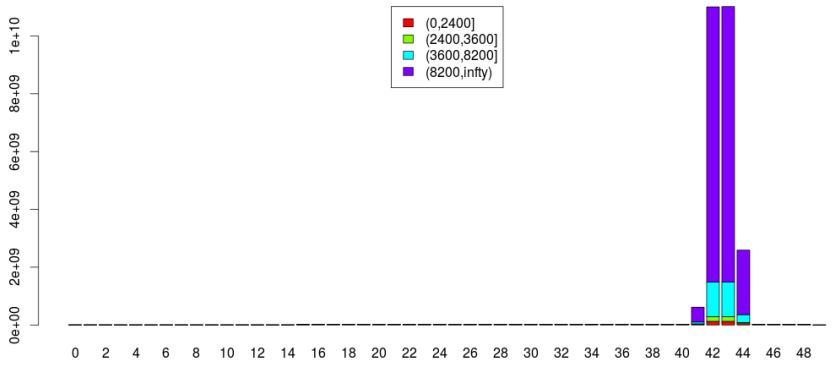

Figure 2: Total size of toys in 4 different size ranges released in a each week.

much is happening, so we are not likely to make really bad decisions, even if we use a very naive approach, and once the action starts, we have all the items and so arrival times do not matter anymore.

3 The basic approach

In this section we discuss the basic approach used in our solution. To squeeze the last couple (million) points we needed to work a bit harder – this is described in the next section. However, the approach described here, if implemented properly, should lead to a score not much larger than $1.27 \cdot 10^9$.

3.1 Speeding up

Speeding up efficiently. We want to speed up in order to produce the largest toys at high ratings, the larger the toy the higher the rating. The question is what constitutes effective usage of small toys. One important realization is that in general we want to produce the speed-up toys as slowly as possible without going over 600 minutes. For example, the best way to use a toy of size 1200 is to produce it at speed 2.0. This is much better than producing it at speed, say, 3.0. This is because in the first case you are accelerating for a longer period of time. Therefore, most of the time we want to attempt using the largest toy we can for speed-up, provided it does fit in the working hours.

Target ratings. Our basic approach was to determine, for each toy, the target rating we want for that toy and try to achieve it by using a sequence of small toys (which we call a speed-up run, or just run). We later abandoned this approach for more complicated solutions, but in some form this was present in our code until the very end.

The rating formula. The target rating formula was of the form $\sqrt{\frac{size}{C}}$, where $C \sim 133\,000$ (chosen experimentally). The reason for using this particular formula is as follows: At first glance it might seem that you want to assign ratings so that each toy runs for more or less the same time. This would lead to a formula like this: $\frac{size}{C}$. However, this is not a good idea. This is because speeding up a slow elf is much more efficient than speeding up a fast elf. If you do the math here, you end up with the square root formula.

The distribution of small toys. If you try doing the above and optimize the constant C, the big toys will run at rating slightly about 0.6. This is not ideal, since we have a decent amount of speed-up toys with sizes like 1 000 or 2 000. These can be used to generate speed-up runs way above this target rating. The solution is to generate speed-up runs that go as far as possible until we run out of those larger toys, and only then settle for the target run formula.

3.2 Things you can do if you are really fast

Once you have these runs that go up to a rating of 4.0 or slightly smaller, you might be tempted to simply assign them to the largest toys. However there are better things to do with them.

The 34h trick. Suppose the rating is exactly 4.0. Instead of producing a very large toy (say of size $22k$), it is in fact better to first work for 34 hours (i.e. two full working days together with the off-time in-between) producing a toy of size $34 \cdot 60 \cdot 4.0 = 8160$, which cuts the productivity down to $4.0 \cdot 1.02^{20} \cdot 0.9^{14} \sim 1.36$, and only then produce a very large toy. This observation also works for ratings slightly smaller than 4.0, and in general you always aim for almost exactly 34 hours and then the final rating is about a third of the initial one.

Oscillations. While we are at a rating of 4.0 or close to it (and before we jump into the 34h trick), we can "clear" the toys that are slightly larger than 2400 (actually we can go up to about 3500) by performing what we call *oscillations*. The basic idea is to repeatedly:

- produce a toy that takes more than a single working day, this reduces the rating, and then
- produce a smaller toy, increasing back the rating.

More complex variants of this idea are also possible, but the basic idea is always similar. During the oscillations, you need to remember that small toys are the real resource here, and not time. You should almost always wait until the next work-day starts before starting a new toy. This is true even if it is, say, 10am.

3.3 First year's mess

The basic approach. Finally, let us tackle the "first year problem". Here the basic approach would be to just let the elves do as much work as possible during that time, without using any speed-up toys, using some sort of naive online algorithm. One should in fact refrain from producing toys with size as large

as 2800, since these toys can be used to speed up the elves during oscillations, even though they cannot be done in one day.

Saving the initial rating. The naive solution presented above is a bit wasteful. As is suggested by Fig. 1 and Fig. 2 and easily verified by direct calculation, there is not enough work for all 900 elves during the first couple of months. Moreover, the elves do not start with the base rating of 0.25, but with a rating of 1.0. Therefore, a better idea is to split elves into two groups. The elves in the first group (whose size we experimentally chose to be 760) produce all available toys of size from the range $[3400, 9000]$. Their productivity immediately drops to 0.25 and stays at this level till the end of this phase. The remaining 140 elves in the second group do not waste their initial 1.0 rating. Instead they wait for the arrival of toys with size between 21500 and 22000, which happens to take place in December 2014, and then use their rating to produce this toy. One note to make here is that we actually refrain from producing small toys up to a size of 3400 and not 2800 as stated earlier. There is a trade-off between this threshold and the size of the second group of elves. The toys with sizes in range $[2700, 3400]$ are handled very efficiently in oscillations anyway. On the other hand, every elf saving his 1.0 rating to produce a large toy presents a rather significant value.

4 The hardcore approach

In this section we discuss the details of our implementation of the ideas presented in the previous section. Note that in many cases we need to depart slightly from these ideas for optimal performance. Also, in some cases, additional non-trivial tools are necessary to implement these ideas.

Before we proceed let us introduce some basic notation. Let $rate(r) = 0.25 \cdot 1.02^{r/60}$ be the speed rating after speeding up for r minutes starting from the minimum productivity of 0.25. Also, let $t(r, s) = \lceil \frac{s}{rate(r)} \rceil$ be the time (in minutes) required to produce a toy of size s, given that the elf first spends r minutes on speeding up.

4.1 Long runs

We first construct runs that reach productivity above $4.0/1.02^{10}$, i.e., at most one day away from productivity 4.0. In our final submission we have used $39k$ such runs (this number has been chosen experimentally). Those runs are constructed by generating an Integer Linear Program (ILP) and solving it in FICO Xpress Optimizer, as follows.

The basic ILP. We represent the current productivity of an elf as an integer from the range $R = [0, 8400]$, which is the number of minutes spent so far on toys production - we assume that all the minutes are during sanctioned hours, which allows for this discretization of the rating space. Note that $rate(8400) \sim 4.0$. We also only consider toys with sizes in $S = \{1, \ldots, 2400\}$ in this stage, since larger toys do not fit in a single day. We design a graph where for a state $r \in R$ and a toy size $s \in S$ we have an directed edge to a state $r + t(r, s)$ (labeled with toy size s), In such a graph we look for an integral flow of size $39k$ from

state 0 to states in $[7801, 8400]$. Instead of specifying capacities for single edges, as is usually done, we demand that the flow must not route more than $cnt(s)$ units through edges labeled with s, where $cnt(s)$ is the number of toys of size s available. Our goal here is to generate $39k$ runs and lose as little speed-up as possible. We measure the speed-up lost by introducing a penalty. For the toy size of s we consider the duration d of its production, as well as the optimal duration $dOpt(s)$ which is $\min(600, d/0.25)$. The penalty for producing toy of size s for d minutes is $dOpt - d$. To summarize, using a toy of size s after speeding up for r minutes incurs a penalty of $pen(r, s) = dOpt(s) - t(r, s)$ We minimize the total penalty over all toys used. This problem can be easily cast as ILP.

$$\begin{aligned}
\text{minimize} \quad & \sum_{r \in R, s \in S} pen(r,s) x_{r,s} \\
\text{subject to} \quad & \sum_{r \in R, s \in S \,:\, r \le 7800 \wedge r + t(r,s) > 7800} x_{r,s} \ge 39\,000 \\
& \sum_{r', s \,:\, r' + t(r',s) = r} x_{r',s} = \sum_{s} x_{r,s} && \forall_{0 < r \le 7800} \\
& \sum_{r \in R} x_{r,s} \le cnt(s) && \forall_{s \in S} \\
& x_{r,s} \in \mathbb{N} && \forall_{r \in R, s \in S}
\end{aligned}$$

Trimming the ILP. First of all, note that even though the usual flow LPs are integral (and so the integrality constraints in a corresponding ILP can be dropped), this is not the case here. The culprits are, of course, the unusual capacity constraints. Moreover, the above ILP is very large. To be able to solve it efficiently we need to reduce its size. To this end, we make an additional assumption, namely we remove all variables $x_{r,s}$ with $pen(r, s) > 10$ (the exact value here was chosen by trial and error), except when $r + t(r, s) \le 600$ (i.e. the toy is produced during the first day of speed up). The reason for this exception is the following: to produce speed-up toys with large sizes on small penalty, we need the right "offset", i.e. remainder modulo 600 (because we minimize the penalty, in a good solution most toys take almost exactly 600 minutes, but for some toys to be produced efficiently we need to be at, say 1534 minutes). So the first day is there to allow the run to obtain a specific offset.

Saving the small toys. There is one last problem with the resulting ILP – it severely overuses toys with very small sizes. We neither have a formal proof of that claim nor did any extensive testing, but we believe this might cause trouble later along the way. Because of that, we impose an additional penalty for every toy of size ≤ 20 used. This penalty is much smaller than the values $pen(r, s)$ so that it only acts as a tie-breaker between same-value solutions.

Extracting the runs. Once the ILP is solved, we perform the standard path decomposition algorithm on the resulting flow. This is particularly simple in our case, since the flow network is acyclic. Here is a typical run resulting from this procedure:

143 2 181 221 269 328 400 488 595 725 884 1077 1313 1601 1952

Note the size 2 toy here. The number of minutes spent on speeding up during the first day is 579, this is not achievable with a single toy. This is the "offset" we need to produce the remaining speed-up toys with duration close to 600 minutes.

4.2 Oscillations

After performing a long run, elf's productivity is high - it is at least 3.28. Now our strategy is to oscillate, i.e., produce toys of medium size (say between 2000 and 3500) and when our productivity drops significantly we rebuild it using toys of size [600,2000]. We extend runs to form oscillations by a local search algorithm which swaps toys between oscillations and tries to form optimal order of an oscillation. Each oscillation also ends with a medium sized toy that takes around 34 hours to produce as mentioned before. The local search works as follows.

Objective function The objective function of the local search is the total gain measured in minutes computed as the difference between total processing time of all toys used, when compared to producing them with productivity 0.25. For example if a toy of size 3 000 is produced at productivity 4.0 it contributes $\lceil 3\,000/0.25 \rceil - \lceil 3\,000/4.0 \rceil$ to the objective function.

At the beginning of the local search for each run we extend it with a greedily made oscillation of size at most 7. This is just to have some solution to start with. Next we try to improve the oscillations with the following moves:

- Removing a toy from an oscillation.
- Adding an unused toy to an oscillation.
- Swapping a pair (or two pairs) of toys between two oscillations.
- Moving a toy from one oscillation to another.
- Swap two oscillations. Recall that they are extending different runs, which means that the productivity at the beginning of oscillations are potentially different.
- Shuffle two oscillations: take all the toys of two oscillations and split them randomly between the two oscillations.

There are three important assumptions we used to control the process, as otherwise vast majority of moves would be leading to much worse states.

1. After any change to an oscillation we try to reorder it optimally. In case of short oscillations (of size at most 4) we try all permutations. Otherwise we try 100 random permutations and pick the best one.
2. For a fixed order of toys in an oscillation we still have to make some choices. In particular we have to decide at which points we should call it a day and wait for the next day 9:00am to avoid going to unsanctioned hours. We do it with a branching brute force algorithm, with additional constraints that if the next toy fits into the current day, we produce it,

whereas if a toy goes into the unsanctioned time for more than 60 minutes, which could be avoided by starting it on the next day, then we call it a day and wait for 9:00am next day.

3. As we noticed that the toys used for 34 hours of work are not a bottleneck resource, we assign them at the very end of an oscillation – they are not swapped by local search rules. When an oscillation is about to be changed by local search, toys of size 4000+ are temporarily removed.

As the set of possible moves is quite large, we introduced additional moves. Even though the new moves are subsumed by the more general moves above, their are helpful in making the convergence faster.

- Swap two toys of similar size (of difference at most 40) between oscillations.
- Replace a toy in an oscillation with a free one of similar size – again the difference in size should be at most 40.

A typical oscillation looks as follows:

$$3403, 648, F, 2004, 3592, 734, F, 755, F, 2137, 2654, F, 3227, 625, F, 2350, F, 2914, F, 7925$$

where F denotes a forced wait till 9:00am next day.

Initially we started with hill climbing, where only improving moves were performed. Later we added more randomization, i.e., doing the so called Metropolis process, where an improving move is always accepted, and if a move makes the objective function smaller by d, then it is accepted with probability exp(-d/TEMP), where TEMP is the temperature of the process. As the process was to slow to perform full simulated annealing, we used constant temperature, and changed it manually when we saw that the worsening moves are accepted too often.

Penalties for small toys When local search optimized oscillations, it was allowed to use any toy of size at least 800 without any restrictions. We also allowed local search to use toys of size $[550, 799]$. However, using each such toy incurred a penalty to the objective function. For each toy from the range $[550, 799]$ we checked experimentally what was the marginal gain from this toy if it would be used by mid runs (described in Section 4.3), and this gain would be considered a penalty when this toy was used in an oscillation. In the end local search used almost all the toys from the range $[600, 799]$.

Assigning runs to elves. We now perform these long runs extended with oscillations. That is we take from a queue the first available elf and assign it a long run with an oscillation, then use toys of size in $[3500, 6000]$ to adjust the offset of the elf (each run has its "perfect" starting offset), and perform the run.

4.3 Mid runs

After all the long runs together with oscillations and final medium and large items are done, there are of course still toys left:

- small toys, with sizes $\leq$ around 600, not used when generating the big runs, and

- large toys, with sizes $\geq$ around 3500, and $\leq$ around $22k$, not produced in the oscillations.

We want to use the small toys to speed up the large ones. We divide this tasks into two subtasks. The first one is using the larger small toys (≥ 151) efficiently. These are the toys that cannot be produced without speeding up first. Here we use an ILP very similar to the one for big runs to generate so called mid runs. The difference is that here we do not impose a hard constraint on the number of runs generated. Instead we formulate the objective function to maximize the total time spent producing the toys with size ≥ 151. The idea is to simultaneously maximize the number of such toys used and minimize the speed-up lost. That it actually works (i.e. optimizes both criteria) we do not have a proof, but it definitely seems to do both rather well. Here is the corresponding ILP:

$$\begin{aligned}
\text{maximize} \quad & \sum_{r \in R, s \geq 151} t(r,s) x_{r,s} & \\
\text{subject to} \quad & \sum_{r', s \,:\, r' + t(r',s) = r} x_{r',s} = \sum_{s} x_{r,s} & \forall_{0 < r < r_{min}} \\
& \sum_{r', s \,:\, r' + t(r',s) = r} x_{r',s} \geq \sum_{s} x_{r,s} & \forall_{r \geq r_{min}} \\
& \sum_{r \in R} x_{r,s} \leq cnt(s) & \forall_{s \in S} \\
& x_{r,s} \in \mathbb{N} & \forall_{r \in R, s \in S}
\end{aligned}$$

Note that the balance constraints now become inequalities instead of equalities. This is because we allow runs of arbitrary lengths. However, we still keep the equality balance constraints for $r < r_{min}$, where $r_{min} = 624$ is the minimum total time of a run containing a toy of size ≥ 151. After all, the goal of this program is to use up all items with sizes ≥ 151.

After this ILP is solved we perform path decomposition of the resulting flow, as in the case of big runs. We also append the biggest available toy at the end of each resulting path, going from longest running path to the smallest one.

4.4 Small runs

The basic ILP. After the midruns are generated, we are left with toys with sizes ≤ 150. To generate the ILP for the runs made of these toys we use a different ILP. We now turn to the actual objective of the problem, which is the total runtime. We use a flow formulation as before, but now we add additional sinks at every productivity r. These sinks let the flow go from node r to a specific large item s. The contribution of such flow to the objective of the ILP is equal to the gain (in minutes) we obtain from producing item s after speeding up for r minutes. Note that this gain is equal to $gain(r,s) = t(0,s) - t(r,s)$.

Below is the resulting ILP.

$$\begin{aligned}
\text{maximize} \quad & \sum_{r\in R, s\in S_B} gain(r,s)x_{r,s} & \\
\text{subject to} \quad & \sum_{r'\in R, s\in S_S \,:\, r'+t(r',s)=r} x_{r',s} = \sum_s x_{r,s} & \forall_{r>0} \\
& \sum_{r\in R} x_{r,s} \le cnt(s) & \forall_{s\in S} \\
& x_{r,s} \in \mathbb{N} & \forall_{r\in R, s\in S}
\end{aligned}$$

We use two disjoint toysets here: S_S contains the small toys, with sizes at most 150, while S_B contains the big ones, with sizes at least 3500. Also, we let $S = S_S \cup S_B$.

Remark: The actual ILP, as generated by `smallruns.cpp`, uses two sets of variables: $x_{r,s}$ and $y_{r,s}$ – one for S_S and another for S_B. However, we decided to use only one here, in order to avoid unnecessarily bloating the ILP.

Trimming the ILP. This program is too large to be solved efficiently, so we reduce the number of sinks by assuming that each large item will be produced at the speed that differs from the $\sqrt{s/C}$ guess that we discussed before by at most δ. This amounts to removing from the program all variables $x_{r,s}$ such that $s \in S_B$ and $|r - \sqrt{s/C}| > \delta$. Reasonable values of δ are between 10 and 100. Smaller δ lead to significantly worse solutions, larger make the program very hard to solve.

Improved target function. In the final version of the program we actually use a slightly more sophisticated target function. Instead of just using a closed form formula like $\sqrt{s/C}$ we assume a fixed marginal utility of small items and then use binary search to find the optimal speed for each big item size (and then use these speed values to sparsify the ILP). More specifically, we fix $marg = 2280$ (chosen experimentally) and for a given size s, we look for a speed rating r such that

$$gain(r + t(r, 60), s) - gain(r, s) \sim t(r, 60) + marg$$

(60 is just a constant here, chosen rather arbitrarily). The LHS of the above equality is the gain from adding an item of size 60 at the end of run with total time r, given that the big item assigned to this run has size s. This should be equal to the assumed marginal gain $marg$. However, crucially, we take into account the fact that adding 60 at the end of the run actually increases its running time by $t(r, 60)$ and this increase contributes to the total running time of the elves. This increase is not accounted for by the $\sqrt{s/C}$ formula, and is the sole reason for introducing the more advanced target function.

Incorporating the midruns. The ILP discussed above generates runs as long as 1450 minutes. On the other hand, some midruns have a length of 600 minutes. This is clearly suboptimal, since these midruns were earlier assigned to the largest available toys. To remedy this, we identify those (midrun,big item) pairs that should be broken up (this is almost equivalent to the running time of the big run being at most 1450, but not quite, see the source code for details).

We then generate additional edges in our flow network corresponding to using a given midrun to produce a specific item. Below is the resulting ILP. We use $mid(r)$ to denote the total number of midruns with running time r.

$$\begin{aligned}
\text{maximize} \quad & \sum_{r\in R, s\in S_B} gain(r,s)(x_{r,s}+z_{r,s}) & \\
\text{subject to} \quad & \sum_{r'\in R, s\in S_S \,:\, r'+t(r',s)=r} x_{r',s} = \sum_s x_{r,s} & \forall_{r>0} \\
& \sum_{r\in R}(x_{r,s}+z_{r,s}) \le cnt(s) & \forall_{s\in S} \\
& \sum_{s\in S_B} z_{r,s} \le mid(r) & \forall_{r\in R} \\
& x_{r,s}, z_{r,s} \in \mathbb{N} & \forall_{r\in R, s\in S}
\end{aligned}$$

Remark: For simplicity we did not incorporate the target function based sparsification in this ILP. This is very easy to do — simply remove the $x_{r,s}$ and $z_{r,s}$ variables with r far from the target speed of s. However, actually including these constraints in the ILP obfuscates the basic ideas unnecessarily.

Size inversions. One funny issue related to this ILP is the following: it is not optimal to produce larger items at higher speed! We only realized it after actually solving the ILPs. Here is an example of what our ILP does:

- run a toy with size 18967 on a 1287 minute speedup, and
- run a toy with size 18874 on a 1289 minute speedup.

One can verify that this is 1 minute better than doing it the other way round because of round-ups, even though we are producing a larger toy slower! This does not influence the final score significantly, but it does a bit.

Why do we need the mid-run ILP? A question one could ask is: why do we need two ILPs, one for mid runs and one for small runs? In principle we do not. We would prefer to put everything in the small runs ILP, as its objective function directly models the problems objective function. Why not do it? The short answer is: the program would be too large. The longer answer:

- There are 1450 nodes in the small runs ILP, if we include the items larger than 150, we need 4200 nodes.
- In the small runs ILP, for every big item we generate sinks that correspond to its target rating $\pm$ a small δ, say 50. The very large items that use long mid runs run way above their target ratings. We do not know how to model this well without blowing up the size of the ILP. This is also exactly why there are no sinks in mid runs ILP, we simply assign runs to items monotonically.

4.5 Gluing the runs – the Euler tour method

Now we are faced with a problem of executing all the small runs and mid runs (each extended with a single large toy), as well as all the moderate size items that are not sped up (with sizes in $[3500, 8000]$). The difficulty here is that each run has some specific time offsets in $[0, 599]$ at which it can be executed with minimal waiting time. If it is run at other offsets this waiting time is much bigger.

Example. Consider this very simple run:

100 171

It is best run starting 400 minutes before the end of work day, i.e. at 12:20. It then requires no waiting. If it is started at 12:10 it requires waiting for 10 minutes after producing the first toy. However, if you start it 12:30, you first need to wait for 6.5h, until the end of the working day. Then you produce the first toy at 9:00 the next day, after which you again need to wait until the end of day. Starting this run at 12:30 is therefore a very bad idea.

Runs and toys as edgesets. We need to split the runs in 900 groups, one for each elf, so that they require more or less equal time, and so that the waiting time is as small as possible. We model this as another ILP. This time we take $V = [0, 599]$ as the nodeset. Each run i corresponds to a set of edges, these are pairs (b, e) that correspond to executing this run with starting offset b and ending offset e. Let $w(i, b)$ be the total waiting time (also called *wait-loss*) necessary when starting the run i at offset b. Also let $W(i) = \min_b w(i, b)$ be the minimum wait possible (also called *forced loss* of i). We only consider b such that $w(i, b) \leq W(i) + tol$, where tol is a wait-loss tolerance parameter (experimentally set to 10). A single toy of size x, that is not a part of any run, can begin at any offset b and end at offset $(b + 4x)\%600$. The choices for these single toys are can be modeled with binary variables $s_{x,b}$.

Euler tours. We now want to choose, for every run and every single toy, a starting offset. Our goal is to do that is such a way that for each offset/vertex o, the total number of runs and toys ending in o is as close as possible to the number of toys and runs starting in o. If we could actually make all offsets perfectly balanced, the resulting graph would have an Euler path. If we get all offset balances close to 0, we can instead decompose the graph into a small number of paths, that we can then glue into a single path. This single path can then be chopped into 900 more or less equal pieces, each corresponding to a single elf.

The ILP. We now present a simplified version of the corresponding ILP. Here $r_{i,b,e}$ corresponds to running run i with starting offset b and ending offset e. Similarly $s_{j,b,e}$ corresponds to running a single item j with starting offset b and ending offset e. In the ILP we are simultaneously minimizing the total imbalance of the graph and the total wait-loss. We (over)estimate that a single unit of imbalance corresponds to 600 minutes lost.

$$\text{minimize } 600 \sum_{o=0}^{599} bal_o + \sum_{i,b,o} w(i,b) x_{i,b,o}$$

subject to

$$bal_o = \left| \sum_{i,b} r_{i,b,o} + \sum_{j,b} s_{j,b,o} - \sum_{i,e} r_{i,o,e} - \sum_{j,e} s_{j,o,e} \right| \qquad \forall_{0 \le o \le 599}$$

$$r_{i,b,e}, s_{j,b,e} \in \{0,1\} \qquad \forall_{b,e,i,j}$$

A couple of notes are in order here:

- This program features an absolute value and is therefore not an ILP. However, it can be made into an ILP using standard techniques, i.e. by changing each equality of the form $bal_o = |RHS_o|$ into two inequalities: $bal_o \ge RHS_o$ and $bal_o \ge -RHS_o$. Note that for this transformation to work, it is crucial that we are minimizing the bal_o variables.
- For every run and toy, the end offset can be computed given a start offset. We use end offsets in the above ILP for clarity only.
- Instead of using a separate variable for every single toy, we can actually group and model them together using a single set of variables (no longer binary). In fact, there are only 150 categories of single toys, depending on the remainder of the toy size modulo 150.
- We can also group identical runs and model them together. This actually decreases the number of variables significantly since the small runs ILP generates large groups of identical runs.
- This ILP is extremely hard to solve, it takes up to 3h to find a reasonable solution!

Alternative formulations. We actually also considered an alternative formulation, where each run can only use a certain (small) set of canonical offsets and the waiting time between the runs is modeled directly using additional edges. It is in theory a stronger formulation and looked simpler. However, it turned out to be even harder to solve for the ILP solver, so we abandoned the idea.

4.6 Equalizing the runs

The pieces found by the Euler tour method described above are roughly equal, but still quite different in sizes. We now assign each piece to one of the elves, compute the finish times for all elves, and use a simple local search algorithm to equalize the tours. This is done by swapping mid size items (they all run at 0.25 rating) with lengths that have the same remainder modulo 150. This does not disturb any offsets and allows us to equalize all elves to within a single day.

4.7 Last touches

Last toys. Each elf also receives a final toy with size $> 20k$ (the sizes of these toys are almost but not exactly equal, since equalizer produces close, but not exactly equal finishing times). These toys can be thought of as running at 2.4, speed because they can run at night with no rest afterwards. We could and should speed up these toys, slightly. We did not do it though.

Reducing the forced loss. The small runs produced by the small runs ILP may sometimes have a rather large forced loss. At the very last moment we implemented a method to slightly reduce this loss by attempting to swap pairs of items in a run and checking whether this improves the total runtime of the run.