COMP526 - Programming Puzzle 1 - Bamboo Cuts

Rob Lamprell - 201459448

Revised Due Date: 23rd April 2021

Introduction

Take the five bamboo plots from the assignment, where each integer represents the growth rate of a particular tree on any given day and provide a periodic schedule to minimize the max-height attained by any single tree in the set. I believe the problem originated from this paper [1]. Notations used are derived from the assignment specifications.

Algorithms

Below are the algorithms developed to provide queues - the implementations can be found in **bambooSolutions.py**:

- **cutMaxHeight()** - cut the bamboo tree with the largest height in each round. More formally, the cuts would be c(t) = i = $\max_{i\in[n]} h_i(t)$. This is a simple and somewhat obvious solution. I believe this is referred to as Reduce-Max in these papers [1, 2], though neither were used to formulate this implementation.
- **cutMaxHeight_minMaxGrowth()** - this is the same as cutMaxheight() with the exception that on tie breakers (2 or more trees with the same max height), c(t) will be chosen based on either the largest (default) or smallest growth rate. This attempts to control the randomness of ordering within the set of bamboo.
- **cutMaxHeight_disallowSequential()** - another take on cutMaxheight(). However, this does not allow sequential cuts of the same bamboo. More formally, c(t)≠c(t-1). If the same tree is selected to be cut on two sequential steps, the second tallest bamboo will be cut instead. This was inspired by investigating the Inequality set's cutMaxHeight() results - the queues produced contain endless rounds where c(t)=0, except after around 2000 cuts, where the queue would alter to [0,1,0,2]. Disallowing the sequential cuts reduces the greediness of the algorithm and encourages exploration of the bamboo plot.
- **cutFrequency()** - calculate a frequency for the bamboo to be cut on. This is obtained by dividing the sum of growth rates by each bamboo's growth rate. Bamboo will be after they pass their respective frequency (threshold). If no tree has passed the threshold, cut the one with the highest growth rate. This was developed in an attempt to wrap the heights and growth rates together. Inspired by the pinwheel scheduling methodology mentioned in these papers [1, 2, 3]
- **cutFrequency()_power(p)** - place the trees into descending order, then calculate the frequency they should be cut in using the formula p^{j+1} (where j is the index post sorting). There is an option to *removeSlack* (enabled by default), which ensures the frequencies add up to 1 - this can sometimes reduce the length of the queue and in when p=2 will cause the final two terms to be equal. Inspiration for this algorithm came from this paper [2].
- **cutEachTreeOnce()** - a completely naive solution. This cuts each tree in the plot once. Intuitively, the best possible solution this can obtain is n*($\max_{i\in[n]} g_i$) - the largest growth rate multiplied by the number of trees in the set.

Lower Bounds (LB)

Two Times Largest Growth Rate

In any set of bamboo n, where n>1, something other than the largest bamboo will need to be cut at some point. This means that we cannot prevent the largest tree from growing to at least twice that of its growth rate. Therefore, our LB must be at least 2*($\max_{i\in[n]} g_i$). Although this should hold in all cases as a lower bound, it may not always be an achievable target - such as when n>2 and all n contain equally large growth rates.

Sum of The Growth Rates

The assignment offers large clue as to another potential LB. This is given by the ratio obtained by max-height/$\sum_{i=0}^{n} g_i$.

We can prove this by contradiction. Let's assume that for a given plot the sum of our growth rates is v. Now assume that there exists a LB, less than v, at (v - 1). On each day, our plot grows by v. But we are now restricted to cutting only v-1. This gives us:

$$vol_t = vol_{t-1} + v - (v - 1)$$

$$v > v - 1$$

$$\therefore vol_t > vol_{t-1}$$

If v is always larger than v-1 then it follows vol_t is always larger than vol_{t-1}. Therefore the solution will tend to infinity. Therefore we have a contradiction - proving the LB cannot be less than the sum of growth rates [1].

Which LB Holds

Simply the largest LB is the one which should be aspired towards. If we cannot have a value smaller than twice the largest growth rate and we cannot have a LB smaller than the sum of growth rates then we choose the largest - which in turn, satisfies both conditions.

Results

Each algorithm has been tested thoroughly using a modified version of the provided test-bed which can be found in the file **bambooSolutions.py**. The test-bed utilised each algorithm to create queues of range [2, 10,000). The queue is treated as an infinitely recurring cycle, which is tested against a run of 10,000 iterations. The max-height from this is recorded. And then the earliest occurrence of the lowest max-height is chosen as a solution. The results of this can be observed in Table-1. Where Lines highlighted in blue denote the best solution(s) found. Finally, a summary of the best max-heights achieved can be found within the file **results.csv**.

Plots

Inequality, Power4 and Factors

Inequality, Power4 and Factors are all able to achieve max-heights equal to two times the largest growth rate of their plots at 4000, 6144 and 24 respectively. These are undoubtedly the best possible max-heights, as without increasing the number of trees cut per day, there will always be a situation where the fastest growing bamboo cannot be cut.

Both Inequality and Power4 have great success across all algorithms - able to achieve their respective LBs of 2*$\max_{i\in[n]} g_i$. However, not all algorithms should be judged equal as some of the solutions created are hundreds or thousands of steps in length. It makes more sense to choose a solution of lower space complexity. For Inequality, these are cutMaxHeight_disallowSequential() and cutFrequency_power(2) - providing queue lengths of 4. It is unlikely there exists a smaller queue than at this max-height, as any solution at queue length 3 would only be able to cut each tree once per cycle. This would mean either h_0 (the first term) would reach 6000, or one of the heights would approach infinity (as it would never be cut). Powers4 also utilises cutFrequency_power(2) to attain it's lowest queue length at 32. It's more difficult to say whether this is the smallest possible queue size.

Factors has more mixed results. Although there are three algorithms which are able to hit the LB value of 2*$\max_{i\in[n]} g_i$ at 24, two of these do so with a queue of length 9997. As only 10,000 iterations were tested it is unclear as to whether this is even a cycle, therefore for the purpose of this test, it cannot be determined if the schedule is periodic and will be disregarded. This leaves cutMaxHeight_disallowSequential() as the best solution attained. It attains this with a queue length of 20. However, this is due to early stopping on the algorithm. In other words, if the algorithm were allowed to run for another 20 cycles, it would not produce the same order-cuts as the previous 20. This gives me pause as to whether it is a good solution.

Fibonacci and Precision.

The the max-height LBs for Fibonacci or Precision are more interesting. For both, the sum of growth rates is larger than 2*$\max_{i\in[n]} g_i$ and should therefore be used as our measure. However, none of the algorithms tested are able to reach this LB. Fibonacci only attains 65 (vs 54) using cutMaxHeight_minMaxGrowth(max) and the queue length of this solution is also the largest of all the bamboo plots at 105. This is the solution I have the least confidence in. I don't believe any of the algorithms have attained the lowest max-height possible.

Finally, although Precision results do not achieve the sum of growth rates LB, it does mange 3*$\max_{i\in[n]} g_i$ and does so consistently with small queue lengths across almost all algorithms. If we accept that the best possible max-height value is 6003, we can use p=3 to create a queue length 3 solution - which is the lowest space complexity possible, at length n. However, as there are n terms, the full cycle obtained from this algorithm is actually length 9. All the algorithms are unable to achieve a max-height smaller than a completely naive solution - cutEachTreeOnce().

Conclusion

Algorithms such as, cutMaxHeight_disallowSequential() and cutFrequency_power() encourage earlier exploration of a given bamboo plot. These less greedy algorithms can provide strong solutions at much lower queue lengths. But, they (and other alogirthms) may not necessarily be able to approach the theoretical LBs of a scheduling problem - if such a solution exists.

Finally, there a likely many more finite solutions which can provide the same or better solutions as defined in Table-1. More research appears to have been undertaken since the original paper, it poses an interesting scheduling problem [2, 3].

Appendix - Table 1

Bamboo Plot	Algorithm	$\sum$ Growth Rates	2*(Max)	Max Height	Ratio (2/3dp)	Queue Length	Algorithm Cycles This Queue
Inequality [2000,1,1]	cutMaxHeight()			4000	1.998	2003	No
	cutMaxHeight_minMaxGrowth(max)			4000	1.998	2003	No
	cutMaxHeight_minMaxGrowth(min)	2002	4000	4000	1.998	2002	No
	cutMaxHeight_disallowSequential()			4000	1.998	4	Yes
	cutFrequency()			4000	1.998	2004	No
	cutFrequency_power(2)			4000	1.998	4	Yes
Power4 [3,12,48,192, 768,3072]	cutMaxHeight()			6144	1.50	1024	No
	cutMaxHeight_minMaxGrowth(max)			6144	1.50	1024	No
	cutMaxHeight_minMaxGrowth(min)	4095	6144	6144	1.50	1028	No
	cutMaxHeight_disallowSequential()			6144	1.50	512	No
	cutFrequency()			6144	1.50	1412	No
	cutFrequency_power(2)			6144	1.50	32	Yes
Factors [1,2,3,4,12]	cutMaxHeight()			24	1.09	~~9997~~	No
	cutMaxHeight_minMaxGrowth(max)			36	1.64	18	No
	cutMaxHeight_minMaxGrowth(min)	22	24	24	1.09	~~9997~~	No
	cutMaxHeight_disallowSequential()			24	1.09	20	No
	cutFrequency()			30	1.36	~~9998~~	No
	cutFrequency_power(2)			32	1.45	16	Yes
Fibonacci [1,1,2,3,5,8, 13,21]	cutMaxHeight()			80	1.48	49	No
	cutMaxHeight_minMaxGrowth(max)			65	1.20	105	No
	cutMaxHeight_minMaxGrowth(min)	54	42	80	1.48	49	No
	cutMaxHeight_disallowSequential)			66	1.22	30	No
	cutFrequency()			78	1.44	~~9997~~	No
	cutFrequency_power(2)			128	2.37	128	Yes
Precision [1000,1999, 2001]	cutMaxHeight()			6003	1.20	5	Yes
	cutMaxHeight_minMaxGrowth(max)			6003	1.20	5	Yes
	cutMaxHeight_minMaxGrowth(min)			6003	1.20	5	Yes
	cutMaxHeight_disallowSequential()	5000	4002	6003	1.20	5	Yes
	cutFrequency()			6003	1.20	6	No
	cutFrequency_power(2)			7996	1.60	4	Yes
	cutFrequency_power(3, False)			6003	1.20	3	No (true cycle length is 9)
	cutEachTreeOnce()			6003	1.20	3	Yes

Table 1: Algorithm Results - Utilising 10k iteration Test-bed - blue highlights the best solution(s) found. Any crossed-out queue lengths suggest the max-height was obtained without a periodic schedule.

The final column describes whether or not the algorithm deviates if ran for longer than the specified queue length. That is to say that an algorithm may provide a queue which solves the solution but, the algorithm itself may not necessarily be able to find which tree to cut at time t.

References

[1] Leszek Gąsieniec, Ralf Klasing, Christos Levcopoulos, Andrzej Lingas, Jie Min, and Tomasz Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In International Conference on Current Trends in Theory and Practice of Informatics, pages 229–240. Springer, 2017.

[2] Davide Bilò, Luciano Gualà, Stefano Leucci, Guido Proietti, and Giacomo Scornavacca. Cutting bamboo down to size. arXiv preprint arXiv:2005.00168, 2020.

[3] Mattia D'Emidio, Gabriele Di Stefano, and Alfredo Navarra. Bamboo garden trimming problem: Priority schedulings. Algorithms, 12(4):74, 2019.