


Week 2

← Week 2



Advanced Problem: Online Advertisement Allocation

Alexander S. Kulikov · Instructor · Week 2 · 4 years ago


Please use this thread to discuss Online Advertisement Allocation advanced problem (make sure to review [forum rules](#) before posting).

0 Upvotes

Reply

Follow this discussion

SUBFORUMS
All
Assignment: Programming Assignment 2

	Earliest	Top	Most Recent
RS	<div>Robert Sizemore · a month ago</div> <p>I gave up on this problem and did the other weeks in the course before coming back to it. Some tips from my experiences (using python3):</p> <p>You can solve this without using numpy, Bland's rule or using a full tableau matrix. I was planning on going as far as I could with vanilla python3 and then porting my code over to vectorized numpy operations when I ran into time limits but I never had to. Although I did run up close to the time limits:</p> <p>Good job! (Max time used: 4.84/6.00, max memory used: 19247104/536870912.)</p> <p>I followed the 2-phase simplex algorithm outlined in "Introduction to Algorithms" (CLRS) and simply updated the same list A with each call to pivot using an extra data structure to track which columns/rows corresponded to which basic/non-basic variables.</p> <p>Although I didn't use Bland's rule, I did periodically check for entering/leaving variables indices in reverse order as per someone else's suggestion.</p> <div><div>0 Upvotes</div><div>Reply</div></div>		
	<div></div> <div>Angel Mora · 3 months ago</div> <p>Given that some have recommended the use of Bland's rule and others say it is not necessary (like me), I put this claims to the test. In my solution I used a hint from Rachel Bowyer: after certain number of iterations I would pick the last column with a negative number. From my previous post, I mentioned the program does this every 15 iterations, but other values can be used.</p> <p>So I re-submitted my solutions with and without Bland's rule and with and without picking the last negative column (reversing in the following). I found it odd that I passed in al four cases, I even double checked the case without Bland's rule and without reversing. These are the results I obtained, where the main difference is in time :</p> <p>Without Bland's rule, without reversing: Good job! (Max time used: 0.99/1.00, max memory used: 10850304/536870912.)</p> <p>Without Bland's rule, with reversing: Good job! (Max time used: 0.02/1.00, max memory used: 10854400/536870912.) [this is the solution that made me pass for the first time]</p> <p>With Bland's rule, without reversing: Good job! (Max time used: 0.96/1.00, max memory used: 10833920/536870912.)</p> <p>With Bland's rule, with reversing: Good job! (Max time used: 0.05/1.00, max memory used: 10842112/536870912.)</p> <p>Clearly not using any of the techniques results in the longest execution time. So this is the base scenario. If to the base escenario we only add Bland's rule there is a slight time improvement (0.99 to 0.96 secs). On the other hand, If to the base case we only add reversing, there is a large change in time (0.99 to 0.02 secs). Now, the usage of both Bland's rule and reversing also has a large effect on the time compared to the base case (0.99 to 0.05) but it is actually better using only reversing without Bland's rule.</p> <p>Of course, these times are the max time used, not the total time (I guess) so there might be cases where using both Bland's rule and reversing results in shorter times. I was using C++ so other languages may have different results. Maybe this is the reason the case without any of the techniques finished so close to the time limit.</p> <p>Finally, I think that reversing has the same or similar effect as Bland's rule. However, since there is proof that it avoids cycling (e.g., in the book Linear Programming with Matlab, section 3.5) I do consider that in practice Bland's has to be used. Likely combined with the technique of reversing.</p> <div><div>1 Upvote</div><div>Reply</div></div>		
MN	<div>Mostafa Nassar · 3 months ago</div> <p>Hello all,</p> <p>I have found that this links may help to solve those two problems with the same code.</p> <p>The Big M Method :</p> <p>https://www.youtube.com/watch?v=upgpVkAkFkQ</p>		

Online Calculator: Simplex Method :

<https://linprog.com/en/main-simplex-method>

<https://cbom.atozmath.com/CBOM/Simplex.aspx?q=sm>

Optimal Diet Problem

c++ (Max time used: 0.00/2.00, max memory used: 9904128/536870912.)

Online Advertisement Allocation

c++ (Max time used: 0.03/1.00, max memory used: 10842112/536870912.)

0 Upvotes Hide 1 Reply



蒋楷文 · 7 days ago · Edited

Hey all! I also pass this problem via C++. And I found these resources very useful (In fact, I just implement Simplex Algorithm based on them).

How it works: <https://www.matem.unam.mx/~omar/math340/std-form.html>, <https://www.matem.unam.mx/~omar/math340/simplex-intro.html>, <https://www.matem.unam.mx/~omar/math340/2-phase.html>.

Why it works: <https://www.matem.unam.mx/~omar/math340/blands-rule.html>

Hope these can help too!

And, there is a very weird thing here. As mentioned in other posts, the program needs to consider the case that the artificial variable is in the basis during the transition from Phase I to Phase II. I do not deal with this situation (not covered in resources I mentioned. But I am still worried), and pass the problem successfully. Is this a coincidence or a law?

Ad Allocation

Good job! (Max time used: 0.02/1.00, max memory used: 10854400/536870912.)

0 Upvotes



Reply

Reply

TY Ting Yang · 4 months ago

Realizing the lecture videos are of limited help in solving this last problem, I started with reading threads in the discussion forum and had an idea that this problem would take a long time for me. I googled a lot, watched several videos, and read many materials posted in this thread or online. Finally, I passed with max time of 0.61s in Python3. Here're some important tips that I think will be useful for someone who gets stuck at this problem and saves much time:

1. About Simplex Method: Although the graphical method mentioned in the lecture videos is intuitive, I find it is hard to implement. And there are very few materials implementing this graphical method. Most of the materials you can find use the so-called Tableau method. If explained well, the tableau method is very easy to understand. I strongly recommend the Linear Programming part in "Algorithms, Part II" on Coursera offered by Princeton University. Within one hour, you will get the basic idea of Simplex Method, and java codes that implement the vanilla Simplex Method. I say it's vanilla, because it only solves the LP problem with feasible solutions.
2. In order to know if LP has feasible solutions (bounded or unbounded), you need Two-phase Simplex Method. Phase I to check if there is a feasible solution. Phase II is to solve the original problem. Thanks to the materials provide by Kevin Mann (AMP-Chapter 02), I understood the concepts and implementation of two-phase simplex method very well.
3. Bland's rule: Someone says Bland's rule doesn't help that much, but I don't think so. It is a very powerful selection criteria to avoid cycling and the grader does have a test case #44 to check this. Wikipedia is enough to know what to do.
4. Tolerance: There are test cases checking if your program is robust to machine rounding errors, too. This means: when checking the result of Phase I, use a small epsilon; when deciding if there is a pivoting column, use epsilon; when calculating ratios for some rows, use epsilon on the coefficient.
5. Transition between Phase I and Phase II: If after Phase I, artificial variables are in the basis, you need to select a non-artificial variable that has a non-zero coefficient in that constraint row. Pivot to replace the artificial variable with this non-artificial variable. Repeat until you get rid of all artificial variables from your basis. Then start Phase II.
6. Time Limit: First you need to make sure your program is correct by passing all the testing cases provided. Then you can work on reducing execution time. The first rule is using numpy and matrix operation whenever you can. This means your tableau should be a matrix, not a list. And all operations are based on matrix. The second rule is using numpy built-in function to find index. Reduce for-loops as much as you can.
7. Other tips: Make sure your enhanced tableau and basis are initialized correctly. Deal with all the coefficients and indices carefully. If time is an issue, then only add artificial variables when the value on the right side is negative. Pivoting in Phase II only cares about non-artificial variables. Phase I objective function includes coefficients of all non-artificial variables corresponding to the constraints where you add artificial variables.

This is all I can think of to pass this problem. One last tip: Don't give up. You know you will nail it!

2 Upvotes Reply





Angel Mora · 5 months ago

This has been the most difficult problem for me among the first 4 courses of the specialization and these two weeks of this fifth course. I do agree with others that the material from the lectures is not enough to solve the problem. So, to summarize some tips that helped me and hoped were all together:

- The two-phase simplex method algorithm that I implemented I found it in Chapter 3 of the book Linear Programming with Matlab. I found it to be the most complete compared to other resources I found in other comments. This chapter explains the two phases and the three cases we are looking for: bounded solution, unbounded solution, and infinity.
- Some suggest to keep in mind to remove the artificial variables in case phase I was used. Although I would also say so, I guess the resources they were using did not make that point very clear. From the chapter I mentioned in my previous point, that was part of the description. So, do remember to remove the artificial variables but this should be clearly said in the resources you are using.
- As many mention, machine precision/floating point arithmetic is VERY important. Do use an epsilon variable (EPS), but do not make it too small or it will not help. Ahmad Bashar Eter suggested not going below 1e-11. I agree as in my tests 1e-12 gave me incorrect results and 1e-11 give me correct results. In fact 1e-10 worked for me too and this is the value I used when I passed.
- There is one test case that I was failing because it was taking too long. For that I did something similar to what Rachel Bowyer suggests. If the method is taking several iterations, I chose the last column as the pivoting column (the variable to enter the basis). I know "several" can mean anything, so in my implementation if the simplex method took more than 15 iterations I would choose the last column. And this would repeat every 15 iterations. 10 and 20 also worked for me, but as I was testing I left it at 15 before submitting.
- Related to the previous point is the use of Bland's rule. I passed without it, so I think it is not necessary. I did implement it and tested it but was getting incorrect results, of course this was because I had some mistakes in other places in my implementation. But the version that I submitted did not use Bland's rule and passed. Maybe it compensated with Rachel's suggestion from my previous point?

Finally, I would suggest to the lecturers to change the "What To Do" section from the problem assignment document to make it clear that in the lectures the simplex method is just mentioned in general terms and it's up to us to find it in detail and implement it. This is completely fine with me as this problem is labeled as "advanced". It is just that the "What To Do" section does not seem to reflect this.

0 Upvotes Reply



Chan Yat Fung · 2 years ago

Pass finally in C++! Really hard problem.. the following is my remark to pass this question.. I mainly struggle with the precision error...

1. Floating-point arithmetic play a importance role.

That is add/minus using double will give a lot of error, it is because of the limitation of machine precision. Two numbers may have same "machine number" but actually they are not equal. So, when add/minus two number, I multiply the numbers by 1.0e7 and do add/minus operation, and after that I divide by 1.0e7 again. That's the main idea I got pass.

Reference: <https://www.codeproject.com/Articles/29637/Five-Tips-for-Floating-Point-Programming>

2. Be careful of remaining artificial variable after phase 1 if you implement two phase method

0 Upvotes Reply

VH

Vince Hartman · 2 years ago

Hi Alexander, all:

I'm having issues with the simplex method cycling for large inputs. I implemented Bland's rule where I take the smallest value in the objective function row of the tableau for the column pivot point. And then with that column, I determine my pivot row point from the smallest ratio of $a[i]/b[i]$ numbers.

My implementation has worked well thus far for the second and the majority of tests on this this third problem. But now with $n,m > 50$, the tableau seems to cycle at times.

Do you have any recommendations or strategies I should review for why it would be cycling?

I've also experimented with taking the smallest index number that is less than 0 for the column pivot point, and that also produced correct results but similarly cycles with large inputs.

0 Upvotes Hide 1 Reply

VH

Vince Hartman · 2 years ago

I was able to solve the problem today. What worked for me for the pivot logic:

For the column pivot, choose the smallest value within the objective function row. And then if there are matches on min, the first column within index order is sufficient. So `min()` function is sufficient for getting the column pivot.

For the row pivot, choose the smallest ratio for $b[i]/a[i]$ where $a[i]$ is greater than 0. If there is a match, choose the row within the smallest *variable* index. So `min()` function of the calculated ratios will not be

sufficient. I had to first get all rows that had the minimum. Than with those rows, get their variables. And then return the row that has the smallest variable.

Lastly, my code performed below the time limit of 6 seconds in python when I submitted it. But my internal stress testing with n,m = 100 had the solution sometimes returning after 20+ seconds at times. I think this suggests that as long as you implement the simplex method and Bland's rule correctly, you'll be within the time frame for this problem. You will only be outside the time frame if your tableau is cycling.

0 Upvotes



Reply

Reply

AR

Anton Rapetov · 2 years ago · Edited

First of all the task is indeed difficult, and it seems the material in the course is not sufficient, at least for me, maybe somebody smart enough can figure out the algo without using external sources, but not me.

So I used the following links to figure out how to implement it:

1. <http://www.matem.unam.mx/omar/math340/index.html>
2. <https://www.zweigmedia.com/RealWorld/tutorialsf4/framesSimplex.html>
3. <https://mat.gsia.cmu.edu/classes/QUANT/NOTES/chap7.pdf>

So for C++ I've got (+- fluctuations):

```
1 Good job! (Max time used: 0.03/1.00, max memory used: 10846208/536870912.)
```

I used regular `double` instead of `long double`, and I used `EPS = 1e-9`. Though when I test my solution against `lp_solver` with random data, I sometimes was getting errors due to precisions.

I tried to test against `Clp`, however I wasn't able to make `Clp` work. :(If somebody can make Clp work, or has prior experience with it, please share your code/knowledge :-)

Here is some insides of how I implemented the task:

1. As I said above, regular `double`, and `EPS = 1e-9`.
2. To find the first feasibility solution I added the artificial variable (the algo is described in the first link above). I've seen several algos that don't require it, and operate in original variable space. E.g.: <https://arxiv.org/pdf/1304.6894.pdf>. But I haven't tested them.
3. I always keep all RHS values except for the objective function positive. Mostly you don't need to do anything for that, except when you create tableau the first time, and after the first special pivot in the phase 1 (The finding feasibility solution phase).
4. As a pivot (during regular maximization) I always chose the column with the highest (by module) negative value in the objective function. And I always chose the one with the smallest index to break a tie. It seems, it worked out well.
5. Plus, when I knew the exact value (e.g.: `1`, or `0`), I always set it directly, I'm not sure, but it could increase precision.

1 Upvote

Reply



Rachel Bowyer · 2 years ago

I think this is probably the hardest problem so far on the course. Most of the algorithms have detailed pseudo code given in the lectures and convenient data structures in the starter files. For this problem you were really on your own.

I implemented the algorithm described in CLRS in Python. It took me quite a lot of study with pen and paper to really get my head around the algorithm.

To prevent cycling, I implemented Bland's rule. To speed up performance on some test cases I chose the pivot column in reverse index order. As assigning indexes to variables is arbitrary then this change should be consistent with Bland's rule. It might be viewed as a bit of a dirty trick (the notorious test #44 only needs a few pivots), on the other hand learning about the sensitivity of performance to choice of pivot was an important learning moment for me.

I had issues getting the correct answers in some cases until I added a EPS to my comparisons.

Even with my Bland's rule trick, I had performance issues. I dropped in numpy and saw my program run at half speed! However, when I vectorized the operations on matrix A, I got enough of a performance boost to get through all the test cases.

Good job! (Max time used: 2.82/6.00, max memory used: 19316736/536870912.)

Given the time limit is roughly twice the time taken by the instructor's program, I will take that :-)

In summary my approach was:

- CLRS algorithm
- Bland's rule to prevent cycling
- Use EPS for accurate comparisons
- Use numpy for matrix A with the pivot operations on A vectorized.

1 Upvote Reply

KM

Kevin Mann · 2 years ago

The best advice I can think of to give on this problem is DON'T GIVE UP. It took me a little over two weeks.

You will probably encounter difficulties with float precision and elapsed time.

Yes, use the two-phase simplex. The first phase is to check if there is a feasible solution.

I would focus on accuracy first, then as you bump up against time limits, rewrite code (or resort to numpy) for additional performance. Note that particular ways of doing things in numpy may have worse performance than vanilla python.

Finding a cogent explanation of simplex is difficult in itself. I found that each source I used seemed to explain the algorithm in subtly different ways or with limiting assumptions that made basic understanding a challenge.

This link was the most understandable and comprehensive that I used: <http://web.mit.edu/15.053/www/AMP-Chapter-02.pdf>. It has slightly broader coverage that what is required for this problem.

Good luck!

2 Upvotes Reply



Tamas Kurics · 2 years ago · Edited

Finally managed to pass the simplex method.... This was a bit hard indeed. I would like to share my experiences with Python, hope some of you will find it useful.

1. Use the two-phase simplex method, since the right hand side b can contain negative values. Most of the online tutorials deal with nonnegative b which can be misleading. So not just slack variables but so called artificial variables should be introduced.
2. Use numpy matrix for storing your tableau and do the required row operations with numpy.
3. Round-off errors have huge impact! I personally used 1e-9 to check non-negativity. Moreover, after the first phase I have replaced all "small" elements with zeros in the tableau to prevent further round-off error propagation. In one of the test examples I failed ("wrong answer"), although my answer and the correct answer coincided (up to the given precision), so I assume that the grader is strict about checking the inequalities $Ax \leq b$, but this problem was solved when I have re-set small values in the tableau to zeros.
4. I have found the most negative element in the objective vector for selecting the pivot column, but if there are more than one of them, pick the one with the largest index.

4 Upvotes Reply

RM

Rafael Marino · 3 years ago

Hi all,

After struggling with the Simplex method for one month. I finally managed to pass the diet problem yesterday. Now I'm stuck on the ad allocation problem.

I implemented the CLRS two-phase simple algorithm in Python3. My implementation is returning correct solutions but it's too slow. I've tried tweaks and removing unnecessary calculations to no avail. I'm still stuck on test case 96/113: Failed case #96/113: time limit exceeded. (Time used: 7.70/6.00, memory used: 10682368/536870912.)

Does anyone have any tips on Python3/CLRS implementation on how to make the code more efficient? I'm using base Python for everything, the only library I'm importing is stdin from sys to read the data.

I'm also using Python's base methods for list/set manipulation: list.index(), list.append(), list.remove(), list.sort(). Someone said to use numpy, but I'm not even sure at which stage to use it.

So close and yet so far :(

Any thoughts are welcome. Thanks.

P.S. For those still struggling. Please don't underestimate the importance of a stress testing program. I took a stress testing program from a kind soul on the forum and adapted it to produce small LPs, then solved manually and found my bugs. For the CLRS implementation, make sure you implement tolerance appropriately on lines 3-12 of simplex. 1e-3 worked fine for me.

0 Upvotes Reply



Ahmad Bashar Eter · 3 years ago

Good job! (Max time used: 0.09/1.00, max memory used: 10694656/536870912.)

Some Tips for C++ implementation:

- Set eps to 1e-11 don't make it smaller than that (I've tried 1e-15 up to 1e-11)
- Use long double with cin for input
- debug your answers on Linux environment. if you are using codeblock on windows then long double is the same as double. if you don't have Linux machine use ideone.com and don't forget to make your code secret.

1 Upvote Reply

KP

Kim Hoan Pham · 3 years ago

Hi guys,

Where can i find the correct pseudocode for 2 phase simplex.



I have found some material so far but it vaguely discussed about the infeasibility in phase 1.

Thank you very much

Best Regards

0 Upvotes Reply

KH Kathryn Hampton · 3 years ago

Does anyone know why the eps value for phase one comparisons between the objective function result and zero needs to be so much bigger than all other comparisons (even those also comparing to zero) ?

0 Upvotes Reply



Reply

Reply