Solution Checker for the Dual Sampling LP

September 13, 2023

The Python script `verifier_sampling.ipynb` can read in a solution to (Dual-Sampling-LP), check its feasibility, and output the objective value. The script can be opened and run by JupyterLab or jupyter-notebook (see `https://jupyter.org`).

The script works with upper and lower bounds for some constraints, thus the solution is not necessarily infeasible if the script reports that some constraint is not met. However, by the choice of the bounds, we can guarantee that the solution is feasible if the script says so.

If the provided solution is infeasible, the script reports the first constraint that is not fulfilled; if the script identifies the solution as feasible, it outputs the objective value as well as an upper bound on the inverse of the objective value. (The inverse is an upper bound on the approximation ratio of the sampling algorithm.)

$$\max\ \sigma^{-2}\cdot y \qquad \text{(Dual-Sampling-LP)}$$

$$\text{subject to}\quad 4\beta\cdot e^{-(i+j-1)\cdot\sigma\beta}\cdot y \ \le\ v_{i,j}+w_{i,j}\quad \text{for } 0\le i\le j\le N \qquad (1)$$

$$(\alpha+\delta_1)\cdot e^{-(k-1)\cdot\sigma\beta}\cdot y+\sum_{j=k}^{N} v_{k,j}+\sum_{j=0}^{k} w_{j,k}+\mathbb{1}_{k=1}\cdot\delta_2\cdot y$$

$$+\mathbb{1}_{k>0}\cdot\left(\sum_{i=1}^{\min\{k,N-k\}} x_{i,k}+\sum_{i=k}^{N-k} x_{k,i}-\sum_{\substack{1\le i\le j\le N,\\ i+j=k}} x_{i,j}\right)\ \le\ e^{-(k+\frac{1}{2})\beta}\qquad \text{for } 0\le k\le N \qquad (2)$$

$$x,y,v,w\ \ge\ 0\ . \qquad (3)$$

1 Input Format

For $\alpha = 1.5$ and $\sigma = 0.663$, the solution is given in the file `sol_sampling_lp.txt`. For $\alpha = 1$ and $\sigma = 0.623$, the solution is given in the file `sol_sampling_lp_alpha1.txt`. In both cases, we choose $N = 2500$ and $\beta = 0.01$.

We encode the solutions using only integers so that the Python script can do exact arithmetic operations instead of floating point operations with unpredictable rounding errors. Since the solution to the LP itself is, of course, not integral, we scale all values of the variables by a factor of 10^{15}.

The first eight lines consist of the values for the parameters `N`, `sigma`, `alpha`, and `beta` (see example below). Even though it is possible to pass solutions with different parameter choices than mentioned above, it is only guaranteed for the above parameter choices that there are no rounding errors due to floating point operations.

The next line consists of the letter `y`, followed by a line containing an integer c, defining $y = c\cdot 10^{-15}$. The next line is `x`, followed by lines that each contain three integers i, j, and c, defining $x_{i,j} = c\cdot 10^{-15}$. The value of a variable $x_{i,j}$ for which the combination of i and j does not appear in the file is implicitly set to 0. After that, there is a block starting with the line `v`, defining the v-variables analogously to x. Finally a block starting with the line `w` defines the w-variables, again in an analogous way.

In the following, we deal with the case $\alpha = 1.5$ and $\sigma = 0.663$. The case $\alpha = 1$ and $\sigma = 0.623$ is handled analogously. As mentioned above, we choose $N = 2500$ and $\beta = 0.01$.

Example

```
N
2500
sigma
0.663
alpha
1.5
beta
0.01
y
142055160369725
x
135 146 1396602044281
135 157 2157996989591
⋮
v
0 1 2651432571617
0 2 5645786984853
⋮
w
0 0 5721148663986
0 1 3031910341277
⋮
259 259 184488084657
```

2 Checking the Constraints

2.1 Checking (1)

Since

$$4\beta \cdot e^{\sigma\beta} \cdot 10^{20} < 4026608108411960334,$$

for $i = j = 0$, it suffices to check that

$$4026608108411960334 \cdot y \leq 10^{20} \cdot (v_{0,0} + w_{0,0})$$

Then we check the other constraints one by one, always maintaining an upper bound on $4\beta \cdot e^{-(i+j-1)\sigma\beta} \cdot 10^{20}$ as follows: Suppose we are given U_{i+j} such that

$$4\beta \cdot e^{-(i+j-1)\sigma\beta} \cdot 10^{20} \leq U_{i+j}$$

(starting with $U_0 = 4026608108411960334$). Then

$$U_{i+j+1} := \lfloor U_{i+j} \cdot 99339192995802757404 \cdot 10^{-20} \rfloor + 1$$

yields an upper bound on $4\beta \cdot e^{-(i+j)\sigma\beta} \cdot 10^{20}$ since $e^{-\sigma\beta} < 99339192995802757404 \cdot 10^{-20}$.

The Python script verifies that $U_{i+j} \cdot y \leq 10^{20} \cdot (v_{i,j} + w_{i,j})$ holds for all $0 \leq i \leq j \leq N$.

2.2 Checking (2)

We always use an upper bound u_k on $(\alpha+\delta_1)\cdot e^{-(k-1)\cdot\sigma\beta}\cdot 10^{20}$. Recall that δ_1 is defined as

$$\delta_1 \;:=\; \frac{4\beta}{e^{N\sigma\beta}(e^{\sigma\beta}-1)}.$$

For $k=0$, we use $(\alpha+\delta_1)\cdot e^{\sigma\beta}\cdot 10^{20} < 150997842396816263968 =: u_0$. We again compute u_{k+1} as

$$u_{k+1} = \lfloor u_k \cdot 99339192995802757404 \cdot 10^{-20} \rfloor + 1.$$

Moreover, we use a lower bound ℓ_k on $e^{-(k+\frac{1}{2})\beta}\cdot 10^{20}$. For $k=0$, we have $e^{-\frac{1}{2}\beta}\cdot 10^{20} > 99501247919268231335 =: \ell_0$, and we can compute ℓ_{k+1} by

$$\ell_{k+1} = \lfloor \ell_k \cdot 990049833749168053 57 \cdot 10^{-20} \rfloor$$

since $e^{-\beta} > 990049833749168053 57\cdot 10^{-20}$.

Recall that δ_2 was defined as

$$\delta_2 \;:=\; \left(\alpha+\;\frac{2\beta}{e^{N\sigma\beta}(e^{\sigma\beta}-1)}\right)\cdot\frac{e^{-N\sigma\beta}}{(1-e^{-\sigma\beta})^2}\cdot(1+N-e^{-\sigma\beta}N)$$

Plugging in $\alpha=1.5$, $N=2500$, $\sigma=0.663$ and $\beta=0.01$, we get

$$\delta_2 < 38110920964373004 49\cdot 10^{-20}.$$

We check the constraints of type (2) for all $0\le k\le N$ using these lower and upper bounds. Note that we always work with variable values that are scaled by 10^{15}. Since the term on the right hand side is a constant, we have to scale it by 10^{15}, too, in order to check the correct inequality.

2.3 Checking (3)

This is straightforward.

3 Computing the Objective Value

The inverse of the objective value gives us an upper bound on the approximation ratio of the sampling algorithm. So we are interested in the value of $\frac{\sigma^2}{y}$. Since $\sigma^2 = 0.439569$,

$$\left(\left\lfloor\frac{439569\cdot 10^{15}}{y\cdot 10^{15}}\right\rfloor+1\right)\cdot 10^{-6}$$

is an upper bound on the approximation ratio of the sampling algorithm.

Running the script on the provided input yields as output an upper bound of $3093794\cdot 10^{-6}$.