TCS Guide of Convex Optimization - Day 4 Homotopy Method

February 2, 2023

1 Basic Property of Linear Programs

- Primal LP $\min_{Ax=b,x\geq 0} c^\top x$ and dual LP $\max_{A^\top y+s=c,s\geq 0} b^\top y$ where $x,s\in\mathbb{R}^n_{\geq 0}$.

Lemma 1. *The duality gap $c^\top x - b^\top y = x^\top s$. In particular, for optimal x,s, we have $x_i s_i = 0$ for all i.*

Lemma 2. *For any positive vector $\mu\in\mathbb{R}^n_{>0}$, there is an unique feasible x and s such that*

$$x_i s_i = \mu_i.$$

2 Interior Point Method and its Basic Properties

2.1 One of the Framework

IPM:

- Invariant: $\frac{1}{2}\cdot t\leq xs\leq \frac{3}{2}\cdot t$. Maintain it via the potential $\Phi(\frac{xs}{t}-1)\stackrel{\text{def}}{=}\sum_i \phi(\frac{xs}{t}-1)\leq C_\phi$.
- Parameter step size $h\approx\frac{1}{100}$.
- Initialize $x=T_x(1)$ and $s=T_s(1)$.
- While $t\geq\frac{\epsilon}{2n}$,
  - If $\Phi(\frac{xs}{t}-1)\geq\frac{C_\phi}{2}$
    * Let $v=-ht\cdot\frac{\nabla\Phi(\frac{xs}{t}-1)}{\|\nabla\Phi(\frac{xs}{t}-1)\|_2}$.
    * Pick $\delta_x,\delta_s,\delta_y$ such that

$$\begin{aligned} S\delta_x+X\delta_s=&v,\\ A\delta_x=&0,\\ A^\top\delta_y+\delta_s=&0.\end{aligned}$$

    * Move $x\leftarrow x+\delta_x$, $s\leftarrow s+\delta_s$
  - $t\leftarrow(1-\frac{h}{2\sqrt{n}})t$.

2.2 Basic Properties

Lemma 3. *(What is the step?) $X^{-1}\delta_x=(I-P)\frac{v}{xs}$. $S^{-1}\delta_s=P\frac{v}{xs}$ where*

$$\begin{aligned} P&=S^{-1}A^\top(AS^{-1}XA^\top)^{-1}AX,\\ v&=-ht\cdot\frac{\nabla\Phi(\frac{xs}{t}-1)}{\|\nabla\Phi(\frac{xs}{t}-1)\|_2}\end{aligned}$$

Lemma 4. *(Is the step feasible?) $\|\delta_x/x\|_2\leq 4h$, $\|\delta_s/s\|_2\leq 4h$.In particular, x,s are always feasible.*

Lemma 5. *(Does the step decrease the potential?)Assume $\phi''(u)\leq O(|\phi'(u)|+1)$ for all u, after the x,s update, we have $\Phi^{(\text{new})}-\Phi\leq\left\langle\nabla\Phi,\frac{v}{t}\right\rangle+O(\|\nabla\Phi\|_2+1)h^2$. In particular, ignoring the h^2 term, Φ is decreasing by $-h\|\nabla\Phi\|_2$.*

3 Robust Interior Point Method

The bottleneck of the algorithm is to solve the equation

$$\begin{pmatrix} S & X & 0 \\ A & 0 & 0 \\ 0 & I & A^\top \end{pmatrix} \begin{pmatrix} \delta_x \\ \delta_s \\ \delta_y \end{pmatrix} = \begin{pmatrix} v \\ 0 \\ 0 \end{pmatrix}.$$

Note that both S and X in the matrix changes slowly, but not exactly sparsely. If we can change a coordinate only when necessary, then we can update the matrix inverse instead of computing it again.

Here is a version of IPM that update the matrix only when necessary:

IPM:

- Invariant: $\frac{1}{2} \cdot t \leq xs \leq \frac{3}{2} \cdot t$. Maintain it via the potential $\Phi(\frac{xs}{t} - 1) \stackrel{\text{def}}{=} \sum_i \phi(\frac{xs}{t} - 1) \leq C_\phi$.
- Parameter step size $h \approx \frac{0.001}{\log n}$, approximation ratio to vectors $\delta \approx \frac{0.001}{\log n}$
- Initialize $x = T_x(1)$ and $s = T_s(1)$.
- While $t \geq \frac{\epsilon}{2n}$,
  - Pick $\overline{x} \in (1 \pm \delta)x$, $\overline{s} \in (1 \pm \delta)s$ and $\overline{t} \in (1 \pm \delta)t$
  - If $\Phi(\frac{\overline{xs}}{\overline{t}} - 1) \geq \frac{C_\phi}{2}$
    * Let $v = -ht \cdot \frac{\nabla\Phi(\frac{\overline{xs}}{\overline{t}} - 1)}{\|\nabla\Phi(\frac{\overline{xs}}{\overline{t}} - 1)\|_2}$.
    * Pick $\delta_x, \delta_s, \delta_y$ such that

$$\begin{aligned} \overline{S}\delta_x + \overline{X}\delta_s =& v, \\ A\delta_x =& 0, \\ A^\top \delta_y + \delta_s =& 0. \end{aligned}$$

    * Move $x \leftarrow x + \delta_x$, $s \leftarrow s + \delta_s$
  - $t \leftarrow (1 - \frac{h}{2\sqrt{n}})t$.

Remark:

- We cannot use the ℓ_2 potential anymore (i.e. we cannot force xs is ℓ_2 close to t). If we only have ℓ_∞ approximation to x and s, we should only hope to force xs is ℓ_∞ close to t.
- Instead, we do

$$\phi(u) = \exp(\lambda u) + \exp(-\lambda u)$$

  for some $\lambda = 10 \log n$ and $C_\phi = 100n$. Note that we still have $\Phi \leq C_\phi$ implies $xs \approx t$ (this is the reason for setting $\lambda = \log n$)
- We still have $\phi''(u) \leq O(|\phi'(u)| + 1)$. (up to log).
- When $100n \geq \Phi \geq 50n$, we have $\|\nabla\Phi\|_2 = \Omega(\sqrt{n})$ and hence

$$\begin{aligned} \Phi^{(\text{new})} - \Phi &\leq \left\langle \nabla\Phi, \frac{v}{t} \right\rangle + \lambda O(\|\nabla\Phi\|_2 + 1)h^2 \\ &\leq \left\langle \nabla\Phi, \frac{v}{t} \right\rangle + \lambda O(\|\nabla\Phi\|_2)h^2. \end{aligned}$$

- The key difference is that v is computed using the approximate vectors. Note that

$$\frac{v}{t} \approx -\frac{h}{\sqrt{n}}\nabla\Phi(\frac{\overline{xs}}{\overline{t}} - 1) \approx -h \cdot \frac{(1 \pm \lambda\delta)\nabla\Phi(\frac{xs}{t} - 1) \pm \lambda\delta}{\sqrt{n}}.$$

  So, we have

$$\Phi^{(\text{new})} - \Phi \leq (-h \pm \lambda\delta h \pm \lambda h^2)\|\nabla\Phi\|_2$$

  If $\delta = \frac{0.001}{\log n}$ and $h = \frac{1}{\log n}$, then we have Φ deceases by almost $h\|\nabla\Phi\|_2$. This finishes the proof for the potential decreases.

4 Discussion

What does this RIPM need? It needs a data structure to maintain x and s as follows:

- Each step, x and s is moved by some linear algebra formula via $\overline{x}, \overline{s}$.
- We only need to know which coordinates of x, s moved a lot, so that we can update $\overline{x}, \overline{s}$.
- We also need to output x, s at the end of the algorithm.

5 Getting n^3 time without fast matrix

In some sense, RIPM reduces solving linear program to a streaming problem of "heavy hitter"-ish. Note that the linear system problem is changing on both the matrix and vector. We can further simplify it by noting:

$$\begin{pmatrix} M & v \\ 0 & -1 \end{pmatrix}^{-1} = \begin{pmatrix} M^{-1} & M^{-1}v \\ 0 & -1 \end{pmatrix}.$$

So, we can view the whole problem simply involves maintaining

$$M^{-1}e_j$$

for some sparsely updating M and a 1-sparse fixed vector e_j. Recall that our matrix M involves terms like

$$x, s, v$$

where v is just a coordinate-wise function of x, s.

Lemma 6. *(Naive maintenance) The cost of maintaining $M^{-1}e_j$ for a sequence of M is bounded by $O(n^2 T)$ where T is the total number of coordinate changes in M.*

Proof. Recall that

$$(M + e_i e_j^\top)^{-1} = M^{-1} + M^{-1}e_i(1 + e_j^\top M^{-1} e_i)^{-1} e_j^\top M^{-1}.$$

Given M^{-1} explicitly, we can compute RHS in n^2 time. Hence, each coordinate update to M takes n^2 time. $\square$

Now, we bound the number of coordinate changes for the following greedy update algorithm:

- For each step, if $\overline{x}_i \notin (1 \pm \delta) x_i$, set $\overline{x}_i = x_i$. (Similar for other variables).

Lemma 7. *(Number of coordinate changes) The greedy update algorithm makes $O(n \log n \log(1/\epsilon))$ changes in total.*

Proof. Let T be the number of changes. Note that every time $\overline{x}$ moves, it moves by at least δ multiplicatively. Hence,

$$\sum_k \| \ln \overline{x}_i^{(k+1)} - \ln \overline{x}_i^{(k)} \|_1 = \Theta(T\delta).$$

Note that the total movement in ℓ_1 of $\overline{x}$ is always lower then $\overline{x}$. Hence, we have

$$\begin{aligned} T\delta &= O(\sum_k \| \ln x_i^{(k+1)} - \ln x_i^{(k)} \|_1) \\ &= O(\sqrt{n} \sum_k \| \ln x_i^{(k+1)} - \ln x_i^{(k)} \|_2) \\ &= O(\sqrt{n} \sum_k \| \frac{x_i^{(k+1)} - x_i^{(k)}}{x_i^{(k)}} \|_2) \\ &= n \log(1/\epsilon). \end{aligned}$$

Hence, we have the bound for T. $\square$

Combining the previous two lemmas, we have a LP algorithm with runtime $\widetilde{O}(n^3 \log(1/\epsilon))$. Note that this does not use fast matrix multiplication. We will discuss next how to use fast matrix multiplication to improve the runtime.

6 Getting $n^{2.38}$ time by batch updates

6.1 Improved update schedule

Idea: Not all update schedules are the same.

Consider two cases:

1. $\sqrt{n}$ step, each step updates $\sqrt{n}$ coordinates.
2. 1 step, each step updates n coordinates.

Which is easier to handle?

The 2nd case because it is batched. It is better for parallel purpose, communication purpose, or can allow us to use fast matrix multiplication.

Question: How to make sure our update is batched?

Idea: Update preemptively to batch the updates.

New update schedule:

- For the k-th step (with k is a multiple of 2^l)
  - If $|\ln x_i^{(k)} - \ln x_i^{(k-2^l)}| \geq \frac{\delta}{2\log n}$, set $\overline{x}_i^{(k)} = x_i^{(k)}$

First, note that we can write

$$x_i^{(l)} - x_i^{(k)} = \sum_{2\log n} x_i^{(????)} - x_i^{(???)}$$

where the ??? in each term different by power of 2. Since our algorithm ensures the error is at most $\delta/2\log n$ for each power of 2 difference, we have that $\overline{x}_i = (1 \pm \delta)x_i$ for all iterations.

Second, we note that

$$\|\ln x_i^{(k)} - \ln x_i^{(k-2^l)}\|_2 \leq h2^l \sim 2^l.$$

Hence, the number of coordinate $\geq \frac{\delta}{2\log n}$ is bounded by

$$\widetilde{O}(\frac{2^{2l}}{\delta^2}) = \widetilde{O}(2^{2l}).$$

One can check within $\sqrt{n}\log(n/\epsilon)$ iterations, the number of changes are still $\widetilde{O}(n)$, but most of the updated happens only few iterations.

Lemma 8. *For the new update schedule, the number of update for iteration k is 2^{2l} if $k = 2^l \times odd$.*

6.2 Improved runtime

Lemma 9. *(Fast matrix multiplication) For the current $\omega = 2.3729$, we know that*

$$T_{n,r,n} = n^2 + n^\omega\sqrt{\frac{r}{n}}$$

where $T_{n,r,n}$ is the cost of multiplying $n \times r$ and $r \times n$ matrix.

Lemma 10. *(Two ways to maintain a matrix) Suppose M and N are matrices with q coordinates different.*

- *Given M^{-1}, we can compute N^{-1} in $O(T_{n,q,n})$ time*
- *Given M^{-1} and $M^{-1}b$, we can compute $N^{-1}b$ in time $O(T_{q,q,q} + nq)$ time.*

Proof. Case 1. Let $N = M + UV$ where $U \in \mathbb{R}^{n\times q}$ and $V \in \mathbb{R}^{q\times n}$. Woodbury matrix identity shows that

$$N^{-1} = M^{-1} - M^{-1}U(I + VM^{-1}U)^{-1}VM^{-1}.$$

The cost of this update rule are

- $M^{-1}U, VM^{-1}, VM^{-1}U$ in no time (because it is just extracting coordinates)

- $(I + VM^{-1}U)^{-1}$ in $T_{q,q,q}$
- $M^{-1}U(I + VM^{-1}U)^{-1}$ in $T_{n,q,q}$
- $M^{-1}U(I + VM^{-1}U)^{-1}VM^{-1}$ in $T_{n,q,n}$.

So, the cost is dominated by the term $T_{n,q,n}$.

Case 2. We have

$$N^{-1}b = M^{-1}b - M^{-1}U(I + VM^{-1}U)^{-1}VM^{-1}b.$$

The cost of this update rule are

- $VM^{-1}b$ in nq
- $(I + VM^{-1}U)^{-1}$ in $T_{q,q,q}$
- $(I + VM^{-1}U)^{-1}VM^{-1}b$ in q^2
- $M^{-1}U(I + VM^{-1}U)^{-1}VM^{-1}b$ in nq.

So, the cost is dominated by the term $T_{q,q,q} + nq$. □

Now, we can write down our algorithm for maintaining the soln:

- For every $n^{2.5-\omega}$,
  - Update the matrix inverse using case 1 in the previous lemma
- Otherwise, maintain the soln using case 2 in the previous lemma

Now, we can bound the cost. Between $n^{2.5-\omega}$ iterations, Lemma 8 shows that there are at most

$$q = n^{5-2\omega} \text{ updates.}$$

So, the total cost of case 2 is

$$\begin{aligned}\sqrt{n}(T_{q,q,q} + nq) =& \sqrt{n}(n^{(5-2\omega)\omega} + nn^{5-2\omega}) \\ \leq& n^2.\end{aligned}$$

So, case 2 is not bottleneck.

For the case 1, the cost is

$$\begin{aligned}&\sum_{t=n^{2.5-\omega},2\times n^{2.5-\omega},2^2\times n^{2.5-\omega},\cdots,\sqrt{n}} \frac{\sqrt{n}}{t} \times T_{n,t^2,n} \\ =&\sum_{t=n^{2.5-\omega},2\times n^{2.5-\omega},2^2\times n^{2.5-\omega},\cdots,\sqrt{n}} \frac{\sqrt{n}}{t} \times (n^2 + n^\omega \frac{t}{\sqrt{n}}) \\ =&\frac{n^{2.5}}{n^{2.5-\omega}} + n^\omega \leq n^\omega.\end{aligned}$$