Seidel's Randomized Linear Programming Algorithm

Kurt Mehlhorn

July 7, 2010

I describe Raimund Seidel's randomized linear programming algorithm [Sei91]. It runs in time $O(d!m)$, where m is the number of constraints and d is the number of variables. When discussing Seidel's algorithm it is customary to use d for the number of variables as d is always used for the dimension in computational geometry and the method has a strong geometric flavor.

We consider the problem of maximizing a linear function $c^T x$ subject to m linear constraints $a_i^T x \leq b_i$, $1 \leq i \leq m$. We also assume for simplicity that we have explicit bounds on the variables, say $-M \leq x_j \leq M$ for $1 \leq j \leq d$. The bound constraints are extra and do not belong to our collection of linear constraints. We formulate the algorithm as a recursive procedure.

LINPROG(m linear constraints in d variables, $d \geq 1$)
if $d = 1$ **then**
  do the obvious and return
end if
if $m = 0$ **then**
  return the appropriate vertex of the bounding box, i.e., set $x_j^* = M$ if $c_j \geq 0$ and $x_j^* = -M$ if $c_j < 0$.
end if
select one of the constraints uniformly at random, say $a^T x \leq b$, and delete it.
call LINPROG recursively on the remaining $m-1$ constraints in d variables; let x^* be the solution returned.
if $a^T x^* \leq b$, i.e., x^* satisfies the constraint removed **then**
  return x^*;
end if
comment: the optimal solution must satisfy $a^T x^* = b$
solve $a^T x = b$ for one of the variables and remove this variable from all constraints.
call LINPROG recursively on the remaining $m-1$ constraints in $d-1$ variables (!!!!).
let $\hat{x}^*$ be the vertex returned. Fill in the value of the variable dropped and return the extended solution.

We have a closer look at the various steps. If $d = 1$, we have a set of m inequalities in a single variable x; some of them bound x from above and some of them bound x from below. If the upper bound is smaller than the lower bound, the problem is infeasible. Otherwise either the

upper or the lower bound is the optimal solution depending on whether c tells us to maximize or to minimize. The base case $d = 1$ takes time $O(m)$.

If $d > 1$ and $m = 0$, we have only the box constraints. We find the optimum solution in time $O(d)$.

If $d > 1$ and $m \geq 1$, we select one the constraints uniformly at random, say $a^T x \leq b$, and remove it. We then call our procedure recursively on the remaining $m-1$ constraints. This takes time $O(d)$ plus $T(d, m-1)$, where $T(d,m)$ is the yet to be determined *expected running time* of our algorithm on m constraints in d variables.

Let x^* be the solution returned by the recursive call. There are two cases: either we are lucky and x^* satisfies the constraint removed or we are unlucky and it does not. In the first case, we simply return x^*. In the latter case, we observe that any optimal solution must satisfy the constraint $a^T x \leq b$ with equality. We solve the equation $a^T x = b$ for one of the variables and then remove the variable from all other constraints by substitution. In this way we obtain $(m-1)+2 = m+1$ constraints in $d-1$ variables; observe that the two bound constraints for the eliminated variable give rise to two new constraints for the $d-1$ remaining variables. We call our procedure recursively on these constraints and obtain a solution $\hat{x}^*$ in $d-1$ variables. We compute the value of the variable dropped and in this way obtain x^* from $\hat{x}^*$.

If we are unlucky, we spend time $O(dm)$ for eliminating a variable, time $T(d-1, m+1)$ for the recursive call, and time $O(d)$ for computing the value of the variable dropped.

What is the probability of being unlucky? If we are unlucky, the constraint removed must be one of the at most d constraints whose removal increases the optimum. Thus the probability of being unlucky is at most d/m. We obtain the following recurrence for $T(d,m)$:

$$T(d,m) = \begin{cases} O(m) & \text{if } d = 1 \\ O(d) & \text{if } d > 1 \text{ and } m = 0 \\ O(d) + T(d, m-1) + \frac{d}{m}\left(O(dm) + T(d-1, m+1)\right) & \text{if } d > 1 \text{ and } m \geq 1 \end{cases}.$$

Theorem 1 $T(d,m) = O(d! \max(1, m-1))$.

Proof: Let C be a sufficiently large constant that covers the big-O terms in the recurrence. We use induction on d and for fixed d induction on m and show $T(d,m) \leq C f(d) \max(1, m-1)$ for a yet to be determined function f.

If $d = 1$, $T(1,m) \leq Cm \leq C f(1) \max(1, m-1)$ provided that $f(1) \geq 2$. Observe that $2\max(1, m-1) \geq m$ for all m.

If $d > 1$ and $m = 0$, $T(d,m) = O(d) \leq Cd \leq C f(d) \max(1, m-1)$ provided that $f(d) \geq d$ for all d.

We come to the induction step. Assume $d > 1$. We first deal with the case $m = 1$ and then

with the case $m > 1$. For $m = 1$,

$$\begin{aligned} T(d,1) &= O(d) + T(d,0) + \frac{d}{m}\left(O(dm) + T(d-1,2)\right) \\ &\le C(d + d + d^2 + d\,f(d-1)\max(1,1)) \\ &\le C(3d^2 + d\,f(d-1)) \\ &\le C\,f(d)\max(1,0) \end{aligned}$$

provided that $f(d) \ge d\,f(d-1) + 3d^2$. For $m > 1$,

$$\begin{aligned} T(d,m) &= O(d) + T(d,m-1) + \frac{d}{m}\left(O(dm) + T(d-1,m+1)\right) \\ &\le Cd + C\,f(d)(m-2) + \frac{d}{m}\left(Cdm + C\,f(d-1)m\right) \\ &\le C(2d^2 + f(d)(m-2) + d\,f(d-1)) \\ &\le C\,f(d)(m-1), \end{aligned}$$

provided that $f(d) \ge d\,f(d-1) + 2d^2$. We define $f(1) = 2$ and $f(d) = d\,f(d-1) + 3d^2$ for $d > 1$. Then

$$\begin{aligned} f(d) &= 3d^2 + d\,f(d-1) \\ &= 3d^2 + d\left(3(d-1)^2 + (d-1)\,f(d-1)\right) \\ &= 3d^2 + d\left(3(d-1)^2 + (d-1)\left(3(d-2)^2 + f(d-2)\right)\right) \\ &= 3(d^2 + d(d-1)^2 + d(d-1)(d-2)^2 + d(d-1)(d-2)(d-3)^2 + d\cdots 2\cdot 1^2) \\ &= 3\cdot d!\left(\frac{d^2}{d!} + \frac{(d-1)^2}{(d-1)!} + \frac{(d-2)^2}{(d-2)!} + \dots\right) \\ &= O(d!) \end{aligned}$$

since the series $\sum_{i\ge 1} i^2/i!$ is converging. ∎

References

[Sei91] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete and Computational Geometry*, 6:423–434, 1991.