On Clarkson's "Las Vegas Algorithms for Linear and Integer Programming When the Dimension is Small"

Robert Bassett

March 10, 2014

Question/Why Do We Care

Motivating Question:

Given a linear or integer program with the condition that the number of variables is small, can we come up with algorithm(s) that are particularly well-suited to solve it efficiently?

Wish list:

- Expected time is polynomial in the number of constraints
- Expected time in terms of dimension isn't "too bad".

The Linear Separability Problem

Given a set X of points in $\mathbb{R}^n$, which is partitioned into two sets U and V, is there a plane that separates the points in U from the points in V?

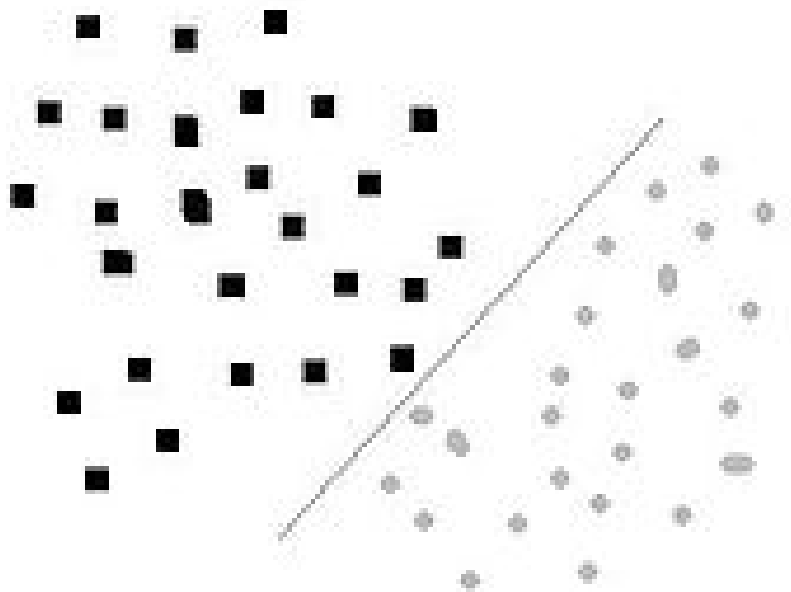

Figure: 2 variables, a whole bunch of restrictions!

Las Vegas Algorithms

A *Las Vegas Algorithm* is a randomized algorithm that always gives correct results.

Las Vegas algorithms either return a solution, or return failure.

First introduced by Laszlo Babai in 1979 in the context of the graph isomorphism problem.

Gamble with the quickness, but not with the result! Hence the name Las Vegas.

Las Vegas Algorithms

A *Las Vegas Algorithm* is a randomized algorithm that always gives correct results.

Las Vegas algorithms either return a solution, or return failure.

First introduced by Laszlo Babai in 1979 in the context of the graph isomorphism problem.

Gamble with the quickness, but not with the result! Hence the name Las Vegas.

Are you feeling lucky?

Spoilers

This paper explores a new algorithm for both linear and integer programs.

For a linear problem with n constraints and d variables, the algorithm requires an expected

$$0(d^2 n) + (\log n) O(d)^{d/2+O(1)} + O(d^4 \sqrt{n} \log n)$$

arithmetic operation.

The paper also presents a related integer programming algorithm with expected

$$O(2^d dn + 8^d \sqrt{n \ln n} \ln n) + d^{O(d)} \phi \ln n$$

operations on numbers with $d^{0(1)}\phi$ bits, as $n \to \infty$

In both of these values, the constant factors do not depend on d or ϕ.

Linear Programming Algorithms

The lay of the land for linear programming algorithms

- Meggido: $O(2^{2^d} n)$
- Dyer and Frieze: Used random sampling to obtain expected $O(d^{3d} n)$ time.
- This Paper: Leading term in n is $O(d^2 n)$.

Note the considerable improvement in d!

The Problem

Determine the maximum x_1 coordinate of points satisfying all contraints, i.e.

$$x_1^* = \max\{x_! | Ax \leq b\}.$$

Where A is an $n \times d$ matrix.

Each inequality in this system defines a closed halfspace H. Call the collection of these halfspaces S. Let $\mathcal{P}(S)$ be the polyhedron defined by S.

It is no loss of generality to assume that the optimum x_1^* is unique, because we can take the x_1^* of minimum 2-norm.

The x_1 of minimum 2-norm is unique, because of the convexity of the solution set

$$\mathcal{P}(S) \cap \{x | x_1 = x_1^*\}.$$

Lastly, this 2-norm mimimizer can be found quickly (Gill, Murray, Wright).

Outline of Algorithm

- $x_s^*(S)$: a simplex like algorithm to solve LPs quickly.
- $x_r^*(S)$: A recursive algorithm designed to quickly throw away redundant constraints.
- $x_i^*(S)$: An iterative algorithm designed to weight the inequalities based on their importance.
- $x_m^*(S)$: A mixed algorithm which combines the first two algorithms. It calls $x_r^*(S)$, and then x_i^*.

x_r^* pseudocode

Function $x_r^*\{S : SetOfHalfspaces\}$

Return $x^* : LPoptimum$

$V \leftarrow \emptyset;\ C_d \leftarrow 9d^2$

If $n \leq C_d$ **Then Return** $x_s^*(S)$

Repeat

choose $R \subset S \setminus V^*$ at random, $|R| = r = d\sqrt{n}$

$x^* \leftarrow x_r^*(R \cup V^*)$

$V \leftarrow \{H \in S | x^* \text{ violates } H\}$

If $|V| \leq 2\sqrt{n}$ **Then** $V^* \leftarrow V^* \cup V$

Until $V = \emptyset$

Return x^*

x_r^* details

The main idea behind x_r^* is the Helly's Theorem

Theorem

Helly Let S be a family of convex subsets of $\mathbb{R}^d$. If the intersection of every $d+1$ of these sets is nonempty, then the whole collection has nonempty intersection.

Applied to our context: The unique optimum of our contraints is determined by d or fewer constraints of S, which we call S^*. (Take the $d+1$th constraint to be $x_1 \geq x_1^*$).

Idea: Randomly choose a subset of planes of a certain size, and optimize the objective over this subset. If this value violates few inequalities in S, the chosen subset is relevant.

x_r^* details

One issue with the x_r^* algorithm is the recursive call.

- If we have a large amount of constraints to be discarded, it may take a long time to randomly sample the "right ones".
- Each of these irrelevant random selections of R calls x_s^*.
- These repeated, unnecessary calls to x_s^* are the bottleneck that slows down x_r^*.

How can we "encourage" the algorithm to choose constraints that are relevant to S^*?

x_i^* pseudocode

Function $x_i^*\{S : SetOfHalfspaces\}$

Return $x^* : LPoptimum$

for $H \in S$ **do** $w_H \leftarrow 1;\ C_d \leftarrow 9d^2$

If $n \leq C_d$**Then Return** $x_s^*(S)$

Repeat

choose $R \subset S$ at random, $|R| = r = C_d$

$x^* \leftarrow x_s^*(R)$

$V \leftarrow \{H \in S | x^*$ violates $H\}$

If $wV \leq 2w(S)/(9d-1)$

Then for $H \in V$ **do** $w_H \leftarrow 2w_H$ **od;**

Until $V = \emptyset$

Return x^*

x_i^* details

This reweighting technique allows us to choose constraints efficiently.

- Choose a random subset R, with probability of selecting a constraint proportional to their weight. Let V be the constraints violated by the optimum by that subset.
- If $|V|$ is low, this subset of inequalities resembles S, so double the weights of these inqualities.

Eventually, the constraints in S^* has large enough weight that the random subset R contains S^*.

Question: Is doubling the weight optimum? Why not triple, or quadruple, the chosen weight? Would this make $S^* \subseteq R$ faster?

The Mixed Algorithm

x_m^* is the mixed algorithm, our "white whale" in terms of complexity.

- It is a version of the recursive algorithm x_r^* that calls the iterative algorithm x_i^* for its recursive calls.
- The motivation for this is to have a time bound with leading term $O(d^2 n)$ while avoiding the large number of calls to x_s^* of the recursive algorithm.

Complexity Analysis

The success of all of these algorithms is based on the following lemma:

Lemma

In the recursive, iterative, and mixed algorithms, if the set V is nonempty, it contains a constraint of S^*.

Draw a picture!

This allows us to be sure that we are making progress with each iteration of the algorithm; elements of S^* are being reweighted.

Complexity Analysis

How often is $|V| \leq 2\sqrt{n}$?

Lemma

Let $V^* \subset S$, and let $R \subset S \setminus V^*$ be a random subset of size r, with $|S \setminus V^*| = n$. Let $V \subset S$ be the set of constraints violated by $x^*(R \cup V^*)$. Then the expected size of V is no more than $d(n - r + 1)/(r - d)$.

Which is used as a tool to prove

Lemma

The probability that any given execution of the loop body has $|V| \leq 2\sqrt{n}$ is at least $1/2$, and so on average two executions are required to obtain a successful one.

Complexity Analysis

Theorem

Given an LP problem with $b \geq 0$, the iterative algorithm x_i^* requires

$$O(d^2 \log n) + (d \log n) O(d)^{d/2+O(1)}$$

expected time, as $n \to \infty$, where the constant factors do not depend on d.

This is broken down into two steps

- The loop body is executed an expected $O(d \log n)$ times.
- The loop body requires $O(dn) + O(d)^{d/2+O(1)}$. This leading term in n, dn, is from the computation that determines V.

Theorem

Complexity Analysis Algorithm x_m^* requires

$$O(d^2 n) + (d^2 \log n) O(d)^{d/2+O(1)} + O(d^4 \sqrt{n} \log n)$$

expected time, as $n \to \infty$, where the constant factors do not depend on d.

Idea of proof: At most $d + 1$ successful ($|V| \leq 2\sqrt{n}$) iterations are needed so that $S \subseteq V^*$.

Take away: $x_i^*(S)$ does pretty well, but by paying really close attention to details we can reduce the leading term in n from $n \log n$ to n.

Integer Programming Algorithms

Lenstra showed that integer linear programming can be solved in polynomial time when the number of variables is fixed.

The fastest deterministic algorithm for this problem requires $d^{O(d)}n\phi$ operations on $d^{O(1)}$ ϕ-bit numbers.

Here ϕ is the *facet complexity* of the input, the maximum number of bits used to specify an input inequality constraint.

Clarkson's algorithm in this paper appeals to Lenstra's algorithm.

The Integer Case

The basis of the algorithm is a result analogous to Helly's Theorem.

Theorem

Doignon-Bell-Scarf There is a set $S^* \subset S$ *with* $|S^*| \leq 2^d - 1$ *and with* $x^*(S) = x^*(S^*)$.

In other words, the optimimum of the ILP is determined by a "small" set.

The Integer Case

The ILP algorithms are simply variations on the LP algorithms

- Sample sizes using 2^d rather than d
- Lenstra's algorithm in the base case
- The set S^* is not necessarily unique. Does Helly's Theorem guarantee uniqueness?

The Integer Case

We have a similar lemma to the linear case

Lemma

Let $V^* \subset S$, and let $R \subset S \setminus V^*$ be a random subset of size $r > 2^{d+1}$, with $|S \setminus V^*| = n$. Let $V \subset S$ be the set of constraints violated by $x^*(R \cup V^*)$. Then with probability $1/2$, $|V| \leq 2^{d+1} n(\ln r)/r$.

The Integer Case

Using this lemma, we have the following modifications for the ILP algorithms

- For the recursive algorithm, put the sample size at $2^d\sqrt{2n\ln n}$
- Use Lenstra's algorithm with $n \leq 2^{2d+5}d$
- With probability $1/2$, the set V will contain no more than $\sqrt{2n\ln n}$ constraints. Require this for a successful iteration.
- For the iterative algorithm, use a sample size of $2^{2d+4}(2d+4)$, with a corresponding $|V|$ bound of $n(\ln 2)/2^{d+3}$.

The Integer Case

Analysis of the ILP mixed algorithm is similar to the LP algorithm

- The top level does expected $2^{d+1}n$ row operations, and generated 2^{d+1} expected subproblems
- Each of these subproblems has no more than $2^{d+1}\sqrt{2n \ln n}$ constraints.
- Each subproblem requires $2^{d+1} \ln n$ iterations.
- Each iteration requires $2^{d+1}\sqrt{2n \ln n}$ row operations, and a call to Lenstra's algorithm.

The Integer Case

These computations yield

Theorem

The ILP algorithm x_m^* requires expected

$$O(2^d n + 8^d \sqrt{n \ln n} \ln n)$$

row operations on $O(d^3\phi)$-bit vectors, and

$$d^{O(d)}\phi \ln n$$

expected operations on $O(d^{O(1)}\phi)$-bit numbers, as $n \to \infty$, where the constant factors do not depend on d or ϕ.

- These row operations are just the the evaluation of an input inequality at a given integral point.
- Clarkson claims that the integral points can be specified with $7d^3\phi$ bits.

The Clarkson Soup

Clarkson-Type Algorithms

Input

- A limit on the number of restrictions necessary to characterize the problem (Helly's theorem, Doignon-Bell-Scarf theorem).
- A way to solve small problems quickly (Simplex-type algorithm, Lenstra's algorithm).
- A metric of how to verify how similar a subproblem is to the original. How many constraints does the solution to this subproblem violate?

Output

Improved complexity results!

The Clarkson Soup

Theme of Clarkson Algorithms:

- Choose a boundary for the metric that guarantees forward progress will be made.
  I.E. If the number of original inequalities that are violated by the randomly selected constraints is small, then the subproblem contains (mostly) necessary constraints of S.
- Choose a sample size so that the probability of this "success" is greater than or equal to $1/2$.
- Do a bunch of random sampling! Outsource small subproblems to whatever the best tool is on the market.

K-feasibility Clarkson

Theorem

DeLoera, Iskander, Louveaux If S is an integer program with k feasible points, there is a subproblem consisting of $c(d, k)$ constraints with the same k-points (and no more).

And so it begins!

We want to use this theorem to establish a Clarkson-type algorithm to find the best k solutions to S.

Given an arbitrary integer program, we can add the constraint $objective_S \geq x_k$, where x_k is the k-th best solution to turn satisfy the conditions for the theorem.

Looking Forward

Open Questions and Ideas:

- How to check for k-feasibility quickly?
  Barvinok's algorithm counts the number of integral points in a polyhedron in polynomial time (in a fixed dimension).
- What kind of metric should be used?
  The number or original constraints violated by the solution to the subproblem will not work, since we aren't interested in the preservation of a single point, but k of them.
  Also, we don't have these k points in our hand like we do for Lensta's and the simplex algorithm!

Violator Spaces

One promising generalization of Clarkson-like-algorithms is Violator Spaces.

A *Violator Space* is a pair (H, V), where H is a finite set and V is a mapping $2^H \to 2^H$ such that

- Consistency: $G \cap V(G) = \emptyset$ holds for all $G \subseteq H$
- Locality: For all $F \subseteq G \subseteq H$, where $G \cap V(F) = \emptyset$.

We think of H as the set of constraints that come with a problem, and the map V is the map that, given a subset G returns the constraints in H that are violated by the optimum over G. (Picture)

Basis and Combinatorial Dimension

We say that $B \subseteq H$ is a *basis* if for all proper subsets $F \subset B$ we have $B \cap V(F) \neq \emptyset$.

In other words, the smallest (by inclusion) set of constraints that have the same optimum objective value.

Let (H, V) be a violator space. The size of a largest basis is called the *combinatorial dimension* of (H, V).

Basis and Combinatorial Dimension

We say that $B \subseteq H$ is a *basis* if for all proper subsets $F \subset B$ we have $B \cap V(F) \neq \emptyset$.

In other words, the smallest (by inclusion) set of constraints that have the same optimum objective value.

Let (H, V) be a violator space. The size of a largest basis is called the *combinatorial dimension* of (H, V).

Helly's Theorem Anybody?!

It turns out that Clarkson's Algorithm holds for generalized violator spaces. (After proving a non-trivial Sampling Lemma.)

Conclusion

Thanks for listening!