09. Linear Programming Generalizations

CPSC 535 ∼ Spring 2019

Kevin A. Wortman

CALIFORNIA STATE UNIVERSITY FULLERTON

April 15, 2019

Big Idea from Algorithm Design: Duality

Duality: When a mathematical object fits two different human models, we can view that object in two ways, (1) *primal* and (2) *dual*.

$\Longrightarrow$ one algorithm could solve two different problems, (1) primal-based and (2) dual-based.

Consequently

- one algorithm actually solves two problems
- two perspectives on the same problem could inspire problem-solving
- reasoning about dual could be easier than primal

Big Idea from Algorithm Design: Parameterized Tractibility

Idea: Sometimes we can overcome efficiency barriers by designing an algorithm that is fast in terms of a parameter that scales up (e.g. n), but slow in terms of a parameter that doesn't in practice (e.g. W).

Recall

- decision sorting must take $\Omega(n \log n)$ time (barrier)
- radix sort takes $\Theta(nW)$ time; faster when $W \in O(1)$ (faster in practice, if not technically faster in theory)

Linear Programming (LP)

- fast (polynomial) on practical inputs
- seems to take worst-case exponential time in theory
- fast pseudopolynomial time i.e. $O((d+n)^{1.5}dW)$
- dimension (# variables) is $O(1)$, fast linear time $O(n)$

Example: lines and points

Primal: set of lines $\{(m, b) : m, b \in \mathbb{R}\}$
Dual: set of 2D points $\{(x, y) : x, y \in \mathbb{R}\}$

Observe: math, and computer, can't distinguish between lines and points! A function that does something to a set of lines, also does something to a set of points.

parallel line search
input: a set $L = \{(m, b) : m, b \in \mathbb{R}\}$ of lines
output: two lines $(m_1, b_1), (m_2, b_2)$ that are parallel, or NIL if no such lines exist

Question: what is the dual of this problem?
Hint: replace "line" $\to$ "point," $m \to x, b \to y$

Duality in the Simplex Algorithm

Idea: view solving an LP in terms of *two* related programs

1. primal LP: original input; goal is to maximize objective function
2. dual LP: minimize "slack" between left-hand-side of inequalities and right-hand-side

Recall: optimal solutions may always be found on simplex vertices

1. in the primal: objective function "pushes" us toward a vertex
2. in the dual: minimizing slack "pulls" us toward an inequality (line/hyperplane)

Standard Form to Slack Form

A standard-form inequality looks like

$$7x_1 + 3x_2 \leq 4$$

Equivalently we may introduce variable s representing the *slack* or difference between the left-hand-side and right-hand-side.

$$7x_1 + 3x_2 + s = 4$$

$$s \geq 0$$

(note that $\leq$ turned into $=$)

Simplex Algorithm (High-Level)

1. Find an initial *feasible solution* inside the simplex of feasible solutions. (Usually trivial.)
2. **repeat**:
   2.1 Find a variable with positive slack in all inequalities.
   2.2 If no such variable exists, **stop**.
   2.3 Eliminate slack; increase var's with positive coefficients, or decrease var's with negative coefficients, until some inequality has zero slack.
   2.4 *Pivot:* This variable/inequality in the primal is maxxed out and cannot be optimized again; exchange it for an inequality/variable in the dual.

Simplex Algorithm Analysis

With d variables and n inequalities, the body of the loop is $\Theta(dn)$.

$\#$ iterations $=$ number of times one variable gets maxxed out before reaching optimal solution (not a function of d, n)

In practice (e.g. last week's exercise), $\#$ iterations is $\sim d, n$ so simplex algorithm takes polynomial time.

Surprise: $\exists$ carefully-crafted LP for which simplex algorithm takes exponential time (e.g. *Klee-Minty cube*)

Worst-case analysis approach is unsatisfying here; can be shown that average runtime is polynomial when averaging over random LPs.

Pseudopolynomial Time Algorithms

After the simplex algorithm was analyzed formally, researchers designed algorithms with provable pseudopolynomial. For word size W, d variables, and n inequalities,

- Khachiyan's *ellipsoid algorithm* takes $O(d^4 W)$ time.
- *Vaidya's algorithm* takes $O((d+n)^{1.5} dW)$ time.
- (there are others)

Mostly of theoretical interest; simplex algorithm remains faster on practical LPs.

Open research questions

1. Is there an LP algorithm that runs in pseudopolynomial time *and* is faster than simplex in practice?
2. Is LP in P? $\Leftrightarrow$ Is there an LP algorithm that runs in strongly polynomial time (function of only d, n, not W)?

Integer Linear Programming

Integer Linear Programming (ILP): same form as LP, but every variable must be an integer, not real number.

maximize $\mathbf{c}^T\mathbf{x}$
subject to $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$
<u>and $\mathbf{x} \in \mathbb{Z}^d$</u>

Mixed Integer Linear Programming (MILP): *some* variables must be integers, others may be any real (integer or not).

Both are *NP*-complete; as far as we know, exponential-time only.

Applications of ILP, MILP

Can use an integer variable to choose between discrete alternatives.

Can introduce indicator variables to decide to use/not use something; e.g. whether an edge is part of a minimum spanning tree.

- Create variable $x \in \mathbb{Z}$
- $0 \leq x$
- $x \leq 1$

Example: Latin Squares

input: $n \times n$ grid with some elements filled with $1, \ldots, n$
output: each element filled with $1, \ldots, n$ such that each value appears exactly once per row and once per column; or NIL if this is impossible

1		
	2	1
2		

$\Longrightarrow$

1	3	2
3	2	1
2	1	3

Latin Squares formulated as ILP

For each row i, column j, value v, each in $1, \ldots, n$, introduce indicator variable

$$g_{i,j,k} = 1 \Leftrightarrow \text{assign value } k \text{ to row } i\text{, col. } j$$

$$\begin{array}{ll}
0 \leq g_{i,j,k} \leq 1 & \forall i, j, k \text{ (each indicator is 0 or 1)} \\
g_{i,j,k} = x & \text{for all pos'n } i, j \text{ pre-filled to } x \\
\sum_j g_{i,j,k} = 1 & \forall i, k \text{ (each value appears once per row)} \\
\sum_i g_{i,j,k} = 1 & \forall j, k \text{ (each value appears once per column)} \\
g_{i,j,k} \in \mathbb{Z} & \forall i, j, k \text{ (indicators are integers)}
\end{array}$$

Low-Dimensional LP

Megiddo: algorithm with runtime

$$O(c^d n)$$

for $c > 1$;

- exponential in d, still counts as exponential time
- **but**, when $d \in O(1)$, this simplifies to

$$O(c^d n) = O((1)n) = O(n)$$

- $\implies$ **linear time** when number of variables d is a small constant

Generalizations of Low-Dimensional LP

More general objective functions still admit $O(c^d n)$-time algorithms

1. *quadratic* programming e.g. maximize $f(x_1, \ldots, x_d) =$ (polynomial of order 2)
2. *cubic* programming e.g. maximize $f(x_1, \ldots, x_d) =$ (polynomial of order 3)
3. *convex* programming e.g. maximize f, a *convex* function
4. *quasiconvex* programming, e.g. maximize f, a *quasiconvex* function

Inequalities are still linear in these frameworks.
Applications include geometric problems (points, lines, circles, etc.).