We express the time to solve a problem instance as a function of the space required to store it.

For more details on this handout, see Chapter 3 of the text by Wright.

1 Storing problem data

Look at the number of bits required to store an instance of a problem. Storing an integer $a$ requires

$$\lceil \log_2 |a| \rceil + 1 \text{ bits.}$$

Rational numbers can be stored by storing the numerator and denominator. Alternatively, for LP, can multiply a constraint by the least common denominator, so we can then assume all data in $A$, $b$, and $c$ are integer. We take $A$ to be $m \times n$, with $b$ and $c$ dimensioned appropriately.

Need to also indicate where each number starts and stops.

So total storage required is denoted by $L$ and is a linear function of the number of entries in the matrix $(mn)$ and the logs of the nonzero data entries themselves. The number $L$ is the length of the data.

2 Rational number model

We work with the rational number model, so we assume all data are rational.

The arithmetic operations of $+,-,\times,\div$ are assumed to take unit time.

For this course, we assume that any square root can be approximated closely enough by a rational in unit time.

**Lemma 1.** (Lemma 3.1, Wright. See text for proof.) When the problem data are integer with length $L$, the vertices of the primal and dual feasible polyhedra defined by

$$\{x \in \mathbb{R}^n : Ax = b, x \geq 0\}, \quad \{(y, s) \in \mathbb{R}^m \times \mathbb{R}^n : A^T y + s = c, s \geq 0\}$$

are rational. Moreover, the nonzero components of $x$ and $s$ for these vertices are bounded below by $2^{-L}$.

**Corollary 1.** Any non-optimal vertex pair $(\bar{x}, \bar{y}, \bar{s})$ has duality gap at least $2^{-2L}$.

**Proof.** Since $(\bar{x}, \bar{y}, \bar{s})$ is non-optimal, it violates complementary slackness. Thus, for some component $i$, both $\bar{x}_i \geq 2^{-L}$ and $\bar{s}_i \geq 2^{-L}$, so the duality gap $\bar{x}^T \bar{s} \geq 2^{-2L}$. 

1
3 Run time

The runtime of an algorithm is the number of arithmetic operations required to determine a solution.

A polynomial algorithm has runtime that depends polynomially on the size of the problem, for any instance of the problem.

An algorithm is strongly polynomial if the number of arithmetic operations it performs is polynomial in the dimension of the problem, independent of the size of the individual entries. For example, Tardos has given an algorithm for linear programming that does not depend on the sizes of the entries in $b$ and $c$ (although it does depend on the sizes of the entries in $A$.) So if we have a class of LPs with bounded entries in $A$ then Tardos’s algorithm is polynomial in just $m$ and $n$, regardless of the entries in $b$ and $c$.

Let $f(n)$ and $g(n)$ be two functions of the integers. If there exist positive constants $c$ and $k$ such that

$$|f(n)| \leq c|g(n)| \quad \text{for all } n \geq k$$

then we say that $f(n)$ is $O(g(n))$. Furthermore, if

$$\frac{f(n)}{g(n)} \to 0 \quad \text{as } n \to \infty$$

then $f(n)$ is $o(g(n))$. 