

Integer programming:

- A Cutting-Plane algorithm for ILP's.
- Branch and Cut for TSP.

Integer programming:

- A Cutting-Plane algorithm for ILP's.
- Branch and Cut for TSP.

Polynomial time algorithms for linear programming:

- The Ellipsoid algorithm.
- Interior point algorithms.

# Cutting Planes

We wish to solve an integer program  $P$ .

# Cutting Planes

We wish to solve an integer program  $P$ .

- Consider the relaxation  $P'$ .
- If the optimal solution to  $P'$  is integral, then it is also an optimal solution to  $P$ .

# Cutting Planes

We wish to solve an integer program  $P$ .

- Consider the relaxation  $P'$ .
- If the optimal solution to  $P'$  is integral, then it is also an optimal solution to  $P$ .

Main idea of cutting planes algorithm:

- Try to obtain the above by adding inequalities to  $P$  that do not change the solution.

# Valid inequalities and Cutting Planes

An inequality is **valid** if it does not change the set of **integer** feasible solutions to  $P$ .

# Valid inequalities and Cutting Planes

An inequality is **valid** if it does not change the set of **integer** feasible solutions to  $P$ .

A valid inequality is a **cutting plane** if it removes the (non-integer) optimal solution to the relaxation  $P'$ .

# A Cutting plane algorithm

Input: ILP  $P$

- 1 Let  $P'$  be the relaxation of  $P$ .
- 2 while optimal solution to  $P'$  is not integral
  - 1 Add valid cutting plane to  $P$ .
  - 2 Let  $P'$  be the relaxation of  $P$ .
- 3 Return optimal solution to  $P'$ .

# A Cutting plane algorithm

Input: ILP  $P$

- 1 Let  $P'$  be the relaxation of  $P$ .
- 2 while optimal solution to  $P'$  is not integral
  - 1 Add valid cutting plane to  $P$ .
  - 2 Let  $P'$  be the relaxation of  $P$ .
- 3 Return optimal solution to  $P'$ .

How to find the valid cutting planes?

# Gomory Cuts

- Assume we solve the relaxation  $P'$  using the simplex algorithm.
- Assume the optimal solution is not integral.

# Gomory Cuts

- Assume we solve the relaxation  $P'$  using the simplex algorithm.
- Assume the optimal solution is not integral.

Consider a line in the final dictionary corresponding to a non-integer variable:

$$x_i = b_i + \sum_j a_j x_j$$

- Assume we solve the relaxation  $P'$  using the simplex algorithm.
- Assume the optimal solution is not integral.

Consider a line in the final dictionary corresponding to a non-integer variable:

$$x_i = b_i + \sum_j a_j x_j$$

Define  $b'_i = b_i - \lfloor b_i \rfloor$  and  $a'_j = a_j - \lfloor a_j \rfloor$ . Then

$$x_i - \lfloor b_i \rfloor - \sum_j \lfloor a_j \rfloor x_j = b'_i + \sum_j a'_j x_j$$

Observations: For integral  $x$ , left hand side is integral, and right hand side is strictly larger than 0.

# Adding a cutting plane

$$x_i - \lfloor b_i \rfloor - \sum_j \lfloor a_j \rfloor x_j = b'_i + \sum_j a'_j x_j$$

Observations: For integral  $x$ , left hand side is integral, and right hand side is strictly larger than 0.

## Adding a cutting plane

$$x_i - \lfloor b_i \rfloor - \sum_j \lfloor a_j \rfloor x_j = b'_i + \sum_j a'_j x_j$$

Observations: For integral  $x$ , left hand side is integral, and right hand side is strictly larger than 0.

Thus the following is a valid cut:

$$x_i - \lfloor b_i \rfloor - \sum_j \lfloor a_j \rfloor x_j \geq 1$$

# Remarks on the algorithm

If the algorithm terminates, it clearly gives correct the output.

# Remarks on the algorithm

If the algorithm terminates, it clearly gives correct the output.

Not guaranteed to terminate! - but can be guaranteed using special pivot rules (non-trivial!)

# Remarks on the algorithm

If the algorithm terminates, it clearly gives correct the output.

Not guaranteed to terminate! - but can be guaranteed using special pivot rules (non-trivial!)

Algorithm is not very practical...

# Cutting Planes algorithms for e.g TSP

- Use ILP formulation of TSP.

# Cutting Planes algorithms for e.g TSP

- Use ILP formulation of TSP.
- Use *specially tailored cutting planes*

# Cutting Planes algorithms for e.g TSP

- Use ILP formulation of TSP.
- Use *specially tailored cutting planes*
- If no further cuts: Branch!

# Example of cut

Simple idea for symmetric TSP:

Every TSP tour has to cross a given cut at least twice.

# Example of cut

Simple idea for symmetric TSP:

Every TSP tour has to cross a given cut at least twice.

Let  $y_{ij} = x_{ij} + x_{ji}$ , where  $x_{ij}$  and  $x_{ji}$  are the integer variables indicating if  $ij$  and  $ji$  are part of the tour.

# Example of cut

Simple idea for symmetric TSP:

Every TSP tour has to cross a given cut at least twice.

Let  $y_{ij} = x_{ij} + x_{ji}$ , where  $x_{ij}$  and  $x_{ji}$  are the integer variables indicating if  $ij$  and  $ji$  are part of the tour.

Constraint:  $\sum_{(i,j) \in S \times T} y_{ij} \geq 2$ .

# Example of cut

Simple idea for symmetric TSP:

Every TSP tour has to cross a given cut at least twice.

Let  $y_{ij} = x_{ij} + x_{ji}$ , where  $x_{ij}$  and  $x_{ji}$  are the integer variables indicating if  $ij$  and  $ji$  are part of the tour.

Constraint:  $\sum_{(i,j) \in S \times T} y_{ij} \geq 2$ .

Given a fractional solution to the relaxation we can find such a violated constraint (if one exists) efficiently using max flow.

# Branch and Cut for TSP

- Branch-and-bound with relaxation being LP-relaxation + set of valid inequalities.

# Branch and Cut for TSP

- Branch-and-bound with relaxation being LP-relaxation + set of valid inequalities.
- When to stop adding inequalities and start branching is a matter of heuristics and experiments.

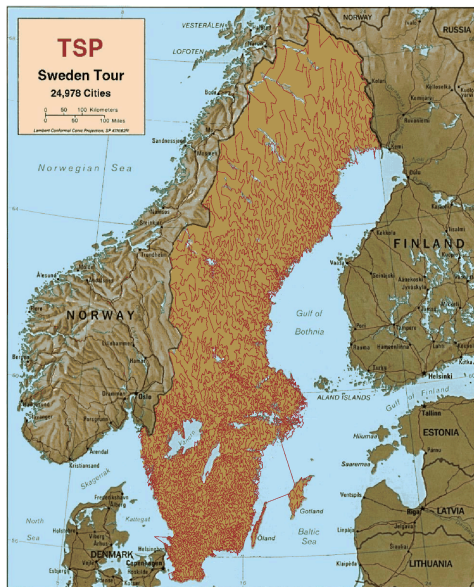
# Branch and Cut for TSP

- Branch-and-bound with relaxation being LP-relaxation + set of valid inequalities.
- When to stop adding inequalities and start branching is a matter of heuristics and experiments.
- Yields state-of-the art solver. Many non-trivial implementation issues.

# State of the art for TSP

- TSP instances of thousand of cities can be consistently solved to optimality using Branch and Cut.
- Instances of up to 25000 cities have been solved: All cities in Sweden!

# TSP tour of Sweden



<http://www.tsp.gatech.edu/sweden/tours/tours.htm>

# Theoretically satisfying LP algorithms

The simplex algorithm runs in worst case exponential time (as a function of the size of the input).

We want an algorithm running in worst case polynomial time (as a function of the size of the input).

# Theoretically satisfying LP algorithms

The simplex algorithm runs in worst case exponential time (as a function of the size of the input).

We want an algorithm running in worst case polynomial time (as a function of the size of the input).

What exactly do we mean by this?

## Model 1:

Our computer holds exact real values.

The input is given by some real matrices. The size of the input is the number of entries in the matrices.

We perform a real arithmetic operation in each step of computation.

The output is the exact real result.

## Model 2:

Our computer holds bits (bytes, words).

The input is given by rational matrices. The size of the input is the total number of bits in the matrices (each number being described in binary notation).

We perform some logical bit operation in each step of computation (or word operations, but we “charge” for each bit operation).

The output is the exact rational result.

# Model 1 vs. Model 2

Model 1 is closer to the way we usually think about algorithms operating with real numbers. It is a very useful abstraction. It is usually easier to design an algorithm for Model 1.

Model 2 is closer to reality. Model 1 cannot be implemented in a 100% faithful way. Model 2 can (and is).

# Model 1 vs. Model 2

Model 1 is closer to the way we usually think about algorithms operating with real numbers. It is a very useful abstraction. It is usually easier to design an algorithm for Model 1.

Model 2 is closer to reality. Model 1 cannot be implemented in a 100% faithful way. Model 2 can (and is).

We know a polynomial time algorithm for linear programming in one of the models but not the other. Which one?

# Model 1 vs. Model 2

Model 1 is closer to the way we usually think about algorithms operating with real numbers. It is a very useful abstraction. It is usually easier to design an algorithm for Model 1.

Model 2 is closer to reality. Model 1 cannot be implemented in a 100% faithful way. Model 2 can (and is).

We know a polynomial time algorithm for linear programming in one of the models but not the other. Which one?

We know a polynomial time algorithm in Model 2 but not in Model 1!

# Do we have time enough to report the output?

For rational instance with an optimal solution, there is a rational optimal solution.

Suppose the input coefficients to an LP instance are rational numbers, given as fractions in binary notation.

Is the number of symbols in the description of an optimal basic feasible solution polynomially bounded in the number of symbols in the input?

# Do we have time enough to report the output?

**Fundamental theorem of Linear Programming:** If there is an optimal solution, there is a basic optimal solution.

**Claim:** The size of the encoding of a basic feasible solution is polynomial in the size of the input.

First: Convert the constraints to constraints with coefficients in  $\mathbf{Z}$ .

Method: If  $p_1/q_1, \dots, p_{n+1}/q_{n+1}$  are the coefficients in the constraint, multiply the constraint by  $q_1 q_2 \dots q_{n+1}$ , or, more practically,  $\text{lcm}(q_1, q_2, \dots, q_{n+1})$

Blowup: At most quadratic - for most practical instances linear.

# Do we have time enough to report the output?

Any basic feasible solution to  $Ax \leq b, x \geq 0$ ,  $A, b$  integer matrices can be obtained by:

a) Introducing slack variables  $x_{n+1}, x_{n+2}, \dots, x_{n+m}$ .

b) Setting  $n$  variables to 0 in the system

$$x_{n+i} = b_i - \sum_{j=1}^n a_{ij}x_j, i = 1..m$$

and solving (uniquely) for the remaining  $m$  variables.

# Do we have time enough to report the output?

Let the resulting system of equations be

$$A'x = b',$$

$$A' \in \mathbf{Z}^{m \times m}, b' \in \mathbf{Z}^m.$$

$A'$  non-singular.

**Cramer's rule:**  $x_j = \det B^{(j)} / \det A'$ , where  $B^{(j)}$  is the result of replacing the  $j$ 'th column of  $A'$  with  $b'$ .

# Do we have time enough to report the output?

Let  $A \in \mathbf{Z}^{m \times m}$  with rows  $a_1, a_2, \dots, a_m$

How big can  $|\det A|$  be?

# Do we have time enough to report the output?

$$\begin{aligned} & |\det A| \\ &= \text{Volume of } \textit{parallelepiped} \text{ spanned by } \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_m \\ &\leq \prod_{i=1}^m \|\mathbf{a}_i\| \\ &= \prod_{i=1}^m \sqrt{\sum_{j=1}^m a_{ij}^2} \\ &\leq \prod_{i=1}^m \sum_{j=1}^m |a_{ij}| \\ &\leq \prod_{i=1}^m (m \max_{j=1}^m |a_{ij}|) \\ &\leq m^m \prod_{i=1}^m (\max_j |a_{ij}|) \end{aligned}$$

# Do we have time enough to report the output?

$$|\det A| \leq m^m \prod_{i=1}^m (\max_j |a_{ij}|)$$

Number of symbols needed to represent  $\det A$  is at most roughly  $m \log m + \sum_i v_i$  where  $v_i$  is length of the representation of the biggest integer in  $i$ 'th row of  $A$ .

Up to constant factors, this is *at most* the number of symbols needed to represent  $A$ .

# We have enough time to report the output!

$x_j = \det B^{(j)} / \det A'$ , where  $B^{(j)}$  is the result of replacing the  $j$ 'th column of  $A'$  with  $b'$ .

Number of symbols necessary to represent each  $x_j$  is up to constant factors at most the number of symbols  $L$  necessary to represent the input.

Total number of symbols necessary to represent solution vector is at most  $O(nL)$ .

# Worst case polynomial time algorithms for LP

1979. Khachian. Ellipsoid algorithm.

- Not practical.
- Does not follow local search pattern.
- **Works by solving the linear inequality problem** (so duality crucial).

1979. Khachian. Ellipsoid algorithm.

- Not practical.
- Does not follow local search pattern.
- **Works by solving the linear inequality problem** (so duality crucial).

1984. Karmakar - Interior Point algorithms

- Many variations. 1990's versions beat simplex.
- Follows local search pattern (with a twist).
- **Duality directly crucial.**

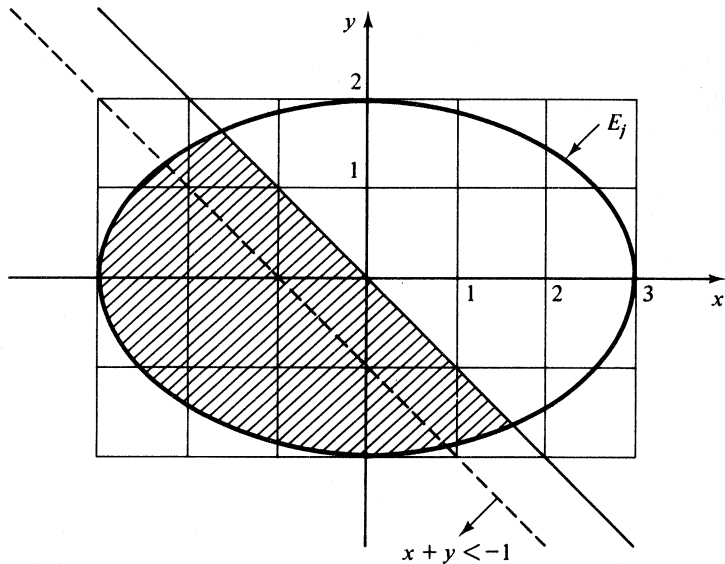
# Ellipsoid Algorithm

We already saw that solving linear programs *reduces* to finding feasible solutions to systems of linear inequalities.

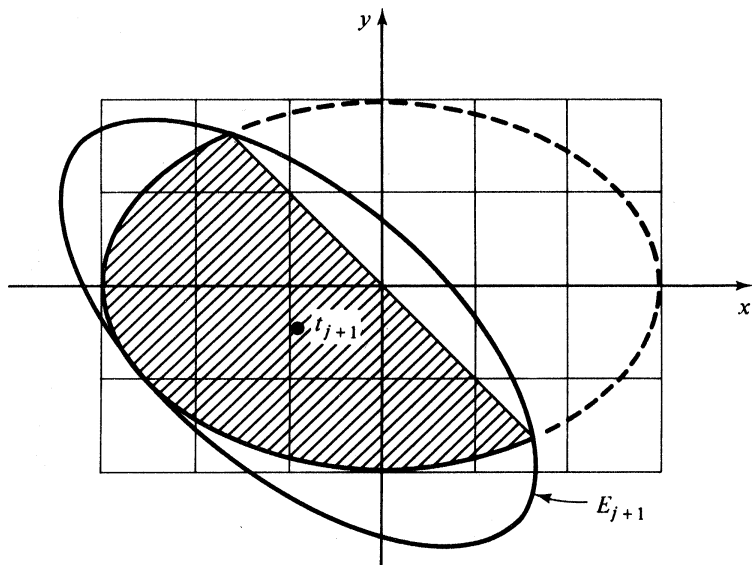
Idea of algorithm:

- Enclose all possible feasible points within a large ball (*i.e.* ellipsoid).
- Test if center of ellipsoid is feasible.
- If not, the ellipsoid is partitioned into two parts by a hyperplane, with all possible feasible points in one part.
- Find a new *smaller* ellipsoid that encloses this half-ellipsoid (pick the smallest), and repeat.

# Finding a half-ellipsoid



# Finding the new ellipsoid



# Why this works

- Each iteration decreases volume by a factor  $2^{-1/2(n+1)}$ .
- First ellipsoid will have volume less than  $(2n^2 2^{2L})^n$ .
- If the system is feasible, the feasible region within the ball  $B(0, n2^L)$  has volume at least  $2^{-(n+2)L}$ .
- Hence at most  $K = 16n(n+1)L$  iterations are needed –  
 $(2n^2 2^{2L})^n 2^{-K/2(n+1)} = (2n^2 2^{2L})^n 2^{-8nL} < 2^{-(n+2)L}$ .

# Why this works

- Each iteration decreases volume by a factor  $2^{-1/2(n+1)}$ .
- First ellipsoid will have volume less than  $(2n^2 2^{2L})^n$ .
- If the system is feasible *and of full dimension*, the feasible region within the ball  $B(0, n2^L)$  has volume at least  $2^{-(n+2)L}$ .
- Hence at most  $K = 16n(n+1)L$  iterations are needed –  $(2n^2 2^{2L})^n 2^{-K/2(n+1)} = (2n^2 2^{2L})^n 2^{-8nL} < 2^{-(n+2)L}$ .

Technical issue: The feasible region may not be of full dimension, and hence can be non-empty but with volume 0.

# Dealing with technical issue

Given system of linear inequalities  $Ax \leq b$ .

Define  $b'_i = b_i + 1/(2^{2L})$  and consider the system of *strict* inequalities  $Ax < b'$ .

# Dealing with technical issue

Given system of linear inequalities  $Ax \leq b$ .

Define  $b'_i = b_i + 1/(2^{2L})$  and consider the system of *strict* inequalities  $Ax < b'$ .

Claim:  $Ax \leq b$  is feasible if and only if  $Ax < b'$  is feasible.

# Dealing with technical issue

Given system of linear inequalities  $Ax \leq b$ .

Define  $b'_i = b_i + 1/(2^{2L})$  and consider the system of *strict* inequalities  $Ax < b'$ .

Claim:  $Ax \leq b$  is feasible if and only if  $Ax < b'$  is feasible.

Clearly, if there is  $x$  such that  $Ax \leq b$  then clearly also  $Ax < b'$ .

# Dealing with technical issue

Given system of linear inequalities  $Ax \leq b$ .

Define  $b'_i = b_i + 1/(2^{2L})$  and consider the system of *strict* inequalities  $Ax < b'$ .

Claim:  $Ax \leq b$  is feasible if and only if  $Ax < b'$  is feasible.

Clearly, if there is  $x$  such that  $Ax \leq b$  then clearly also  $Ax < b'$ .

On the other hand one can easily modify a solution  $x'$  to  $Ax' < b'$  into another solution  $x$  to  $Ax \leq b$ .

Intuition:  $x'$  is so close to a (unique) basic feasible solution  $x$  that we can easily identify the  $x$ .

An ellipsoid in  $R^n$  is defined as follows:

- Let  $Q$  be a nonsingular  $n \times n$  matrix, and let  $t$  be a vector.

An ellipsoid in  $R^n$  is defined as follows:

- Let  $Q$  be a nonsingular  $n \times n$  matrix, and let  $t$  be a vector.
- This gives affine transformation  $T(x) = Qx + t$ .

An ellipsoid in  $R^n$  is defined as follows:

- Let  $Q$  be a nonsingular  $n \times n$  matrix, and let  $t$  be a vector.
- This gives affine transformation  $T(x) = Qx + t$ .
- The ellipsoid determined by  $T$  is the image  $T(S_n)$  of the unit sphere  $S_n = \{x \mid x^T x \leq 1\}$ .

An ellipsoid in  $R^n$  is defined as follows:

- Let  $Q$  be a nonsingular  $n \times n$  matrix, and let  $t$  be a vector.
- This gives affine transformation  $T(x) = Qx + t$ .
- The ellipsoid determined by  $T$  is the image  $T(S_n)$  of the unit sphere  $S_n = \{x \mid x^T x \leq 1\}$ .
- Thus  $T(S_n) = \{x \mid (x - t)^T Q^T Q(x - t) \leq 1\}$ .

A matrix of the form  $B = QQ^T$  is a *positive definite* matrix.

# The Algorithm

Input: System  $Ax < b$ .

- 1 Let  $t_0 := 0$ ,  $B_0 := n^2 2^{2L} I$ .
- 2 For  $j := 1$  to  $K = 16n(n+1)L$  do
  - 1 (Current ellipsoid is  $E_j = \{x \mid (x - t_j)^T B_j^{-1} (x - t_j) \leq 1\}$ ).
  - 2 If  $At_j < b$ , Return  $t_j$ .
  - 3 Otherwise, find an inequality  $a_i^T t_j \geq b_i$ .
  - 4 Let  $t_{j+1} := t_j - \frac{1}{n+1} \frac{B_j a_i}{\sqrt{a_i^T B_j a_i}}$ .
  - 5 and  $B_{j+1} := \frac{n^2}{n^2-1} \left[ B_j - \frac{2}{n+1} \frac{(B_j a_i)(B_j a_i)^T}{a_i^T B_j a_i} \right]$ .
- 3 Return "Infeasible".

The algorithm involves taking a squareroot!

The algorithm involves taking a squareroot!

Solution:

- Compute with limited precision.
- In the analysis make each new ellipsoid slightly bigger to account for introduced errors.
- Double the number of iterations to account for larger volumes.

# Unique feature of the Ellipsoid Algorithm

We do not need an explicit description of the system of inequalities:  
We just need an efficient *separation oracle*.

# Unique feature of the Ellipsoid Algorithm

We do not need an explicit description of the system of inequalities:

We just need an efficient *separation oracle*.

We can potentially solve systems with *exponentially* many inequalities!

This is utilized in many applications of the Ellipsoid method.

# Interior Point Algorithms - Intuition

A polyhedron has complex structure on the border with exponentially many corners, edges, etc. Navigating there takes time.

Well inside the polyhedron there is no structure. One can navigate *directly* towards the optimum point without obstacles.

# Local Search Pattern

```
LocalSearch(ProblemInstance  $p$ ){  
   $x :=$  feasible solution to  $p$ ;  
  while( $\exists y \in N(x) : y$  better than  $x$ )  
     $x := y$ ;  
  return  $x$ ;  
}
```

## Design:

- 1 How to find first feasible solution.
- 2 Define neighborhood structure  $N$ .
- 3 Strategy for choosing neighbor.

## Analysis:

- 1 Partial correctness (Termination  $\Rightarrow$  Correctness).
- 2 Termination.
- 3 Complexity.

# Ye's interior point algorithm

**Primal P:** Minimize  $c^T x$  so that  $Ax = b$ ,  $x \geq 0$ .

# Ye's interior point algorithm

**Primal** P: Minimize  $c^T x$  so that  $Ax = b$ ,  $x \geq 0$ .

**Dual** D: Maximize  $b^T y$  so that  $A^T y \leq c$

# Ye's interior point algorithm

**Primal** P: Minimize  $c^T x$  so that  $Ax = b$ ,  $x \geq 0$ .

**Dual** D: Maximize  $b^T y$  so that  $A^T y \leq c$

Equivalent D: Maximize  $b^T y$  so that  $A^T y + s = c$ ,  $s \geq 0$

# Ye's interior point algorithm

**Primal**  $P$ : Minimize  $c^T x$  so that  $Ax = b$ ,  $x \geq 0$ .

**Dual**  $D$ : Maximize  $b^T y$  so that  $A^T y \leq c$

Equivalent  $D$ : Maximize  $b^T y$  so that  $A^T y + s = c$ ,  $s \geq 0$

Ye's algorithm maintains solution  $x^*$  to  $P$  and solution  $y^*, s^*$  to  $D$  *simultaneously*.

# Objectives

We want to stay *well within* the polyhedron  $\Rightarrow$

We modify the cost function to reward staying well within the polyhedron:  
Small  $x$ , small  $s$  is *bad*.

We want to stay *well within* the polyhedron  $\Rightarrow$

We modify the cost function to reward staying well within the polyhedron:  
Small  $x$ , small  $s$  is *bad*.

**Problem:** In the final basic optimal solution, we will be at a corner, so we should not penalize too heavily!

# New objective function (the potential function)

$$G(x, y, s) = (n + \sqrt{n}) \ln(c^T x - b^T y) - \sum_{j=1}^n \ln(x_j s_j)$$

$c^T x - b^T y$  is the *duality gap*. For optimum  $x, y$  it is zero and hence its log is  $-\infty$ .

$-\sum_{j=1}^n \ln(x_j s_j)$  becomes  $\infty$  when one of the  $x_j$ 's or  $s_j$ 's becomes 0.

Near the optimum  $x$  and  $y, s$ , the first term “wins”.

An input instance can be transformed into another instance where we can easily find  $x^*, y^*, s^*$  so that

$$G(x^*, y^*, s^*) = O(\sqrt{nL})$$

and so that an optimum solution to the new instance yields solving the original instance.

# Gradient descent

Gradient descent is a local search technique for maximizing real-valued function  $f$  defined on  $\mathbf{R}^n$ .

Improve( $x^*$ ):

Compute *gradient*  $g = \nabla f(x^*) = (\partial f / \partial x_1, \dots, \partial f / \partial x_n)(x^*)$ .

Let  $x^* = x^* + \delta g$  for small  $\delta > 0$ .

# Gradient Descent in Ye's algorithm

Given current solution  $x^*, y^*, s^*$ . Compute the projection  $d$  of  $\nabla_x G(x^*, y^*, s^*)$  on  $\{x \mid Ax = b\}$ .

# Gradient Descent in Ye's algorithm

Given current solution  $x^*, y^*, s^*$ . Compute the projection  $d$  of  $\nabla_x G(x^*, y^*, s^*)$  on  $\{x \mid Ax = b\}$ .

If  $\|d\| \geq 0.4$ , we let the new solution be  $x' = x^* - \frac{1}{4\|d\|}d$  (a *primal* move)

# Gradient Descent in Ye's algorithm

Given current solution  $x^*, y^*, s^*$ . Compute the projection  $d$  of  $\nabla_x G(x^*, y^*, s^*)$  on  $\{x \mid Ax = b\}$ .

If  $\|d\| \geq 0.4$ , we let the new solution be  $x' = x^* - \frac{1}{4\|d\|}d$  (a *primal* move)

If  $\|d\| < 0.4$  we make a similar gradient descent step on  $y^*, s^*$  rather than  $x^*$ . (a *dual* move)

# Gradient Descent in Ye's algorithm

Given current solution  $x^*, y^*, s^*$ . Compute the projection  $d$  of  $\nabla_x G(x^*, y^*, s^*)$  on  $\{x \mid Ax = b\}$ .

If  $\|d\| \geq 0.4$ , we let the new solution be  $x' = x^* - \frac{1}{4\|d\|}d$  (a *primal* move)

If  $\|d\| < 0.4$  we make a similar gradient descent step on  $y^*, s^*$  rather than  $x^*$ . (a *dual* move)

Calculations show:  $G$  decreases by at least  $\frac{7}{120}$

# When do we stop?

$$G(x, y, s) = (n + \sqrt{n}) \ln(c^T x - b^T y) - \sum_{j=1}^n \ln(x_j s_j)$$

# When do we stop?

$$G(x, y, s) = (n + \sqrt{n}) \ln(c^T x - b^T y) - \sum_{j=1}^n \ln(x_j s_j)$$

**Practical Answer:** When  $G(x, y, s)$  is very small.

$$G(x, y, s) \leq -k\sqrt{n}L \Rightarrow c^T x - b^T y \leq e^{-kL}$$

Worst case number of iterations before halting:  $O(\sqrt{n}L)$

# When do we stop?

**Theoretical Answer:** We must output the exact rational answer.

If  $c^T x - b^T y < 2^{-2L}$  then any basic feasible solution with cost smaller than or equal to the cost of  $x$  is optimal.

An auxiliary procedure can be used to “round” such  $x$  to the desired basic feasible solution. *Before this final rounding, doing limited precision arithmetic is okay!*

Worst case number of iterations before halting:  $O(\sqrt{nL})$

**The ellipsoid and interior point algorithms exactly solves LP over  $\mathbb{Q}$  in using a number of bit operations that is polynomial in  $L$ , the number of symbols in the input (Model 2).**

**The ellipsoid and interior point algorithms exactly solves LP over  $\mathbf{Q}$  in using a number of bit operations that is polynomial in  $L$ , the number of symbols in the input (Model 2).**

**Open problem:** Is there an “exact real” algorithm that solves LP over  $\mathbf{R}$  using polynomial in  $m, n$  many arithmetic operations (Model 1)? Such an algorithm is called **strongly polynomial**. It is very likely that some version of simplex will work!