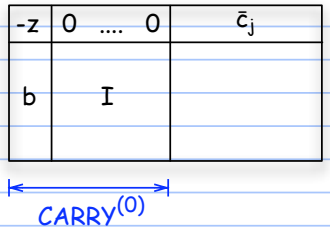
5.1 The revised simplex algorithm

- The full tableau is redundant (identity matrix!) and may contain very many non-basic columns (all of them are stored at every pivot step, => much memory for every tableau). The revised simplex algorithm uses in every step only essential information in a memory-efficient way.
  - The core of that information is the inverse B^{-1} of the current basis B, from which we can easily compute all information needed for a pivot step
  - The revised simplex algorithm is used in all commercial LP-codes

- Main idea of the revised simplex algorithm: the CARRY matrix
  - consider the initial tableau with identity matrix = initial basis on the left in the tableau



because of (4.13), matrix I will change to the inverse B^{-1} of the current basis B in subsequent pivot steps

after iteration ℓ the tableau contains the following data in den first m+1 columns

$-z'$	$-\pi^T$	
b'	B^{-1}	

$\leftarrow$ CARRY$^{(\ell)}$ $\rightarrow$

where

- π^T = dual solution because of (4.12), in general infeasible.

  The numbers π_i are also called simplex multipliers.
- $b' = B^{-1}b$ is the current right hand side = current primal solution
- $z' = c_B{}^T B^{-1} b$ is the current primal cost

It suffices to maintain the following data for the simplex algorithm

1. initial tableau

	c^T
b	A

2. current CARRY-matrix CARRY$^{(\ell)}$
3. current basis by its column indices B(1),...,B(m)

From 1., 2., and 3., one can obtain all data required for a pivot step

(1) Pricing Operation (computing reduced costs)

iteratively compute

$$\bar{c}_j = c_j - \pi^T A_j$$

for non-basic variables until some reduced cost $\bar{c}_j < 0$ or $\bar{c} \geq 0$ (=> termination with an optimal solution)

(2) Generation of the Pivot Column (transforming the pivot column w.r.t. the current basis)

compute

$$X_s = B^{-1} A_s$$

= column s of the current tableau X

the pivot element x_{rs} is obtained as

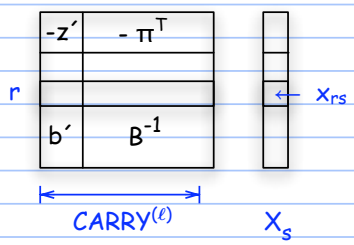
$$\min_{i,\, x_{is} > 0} \frac{b'_i}{x_{is}} \quad \leftarrow \text{ in } CARRY^{(\ell)} \quad \leftarrow \text{ in } X_s$$

or z is unbounded is (if all $x_{is} \le 0$)

(3) Pivot Operation (pivot step)

compute $CARRY^{(\ell+1)}$

i.e., transform X_s into the unit vector (with 1 at the pivot element) and apply the corresponding row operations to $CARRY^{(\ell)}$



(4) Basis Update

set B(r) := s

The Two-Phase-Method works similarly

one starts with the artificial cost vector

(1, ..., 1, 0, ..., 0)

artificial variables

$x_1^a, \ldots, x_m^a$ form the initial basis

Transformation to reduced cost w.r.t. this basis changes the cost to

$$d_j := -\sum_{i=1}^{m} a_{ij}$$

for each non-basic variable

At the end of Phase I we change over to the original cost vector

=> compute $-\pi^T = -c_B^T B^{-1}$ and put it into $CARRY^{(\ell)}$

compute $-z = -c_B^T b'$ and put it into $CARRY^{(\ell)}$

The required data is available: c_B is stored in the initial data, B is stored, and B^{-1} and b´ are stored in

$CARRY^{(\ell)}$

(1) Do not look at every non-basic columns per iteration

one needs all reduced costs

$$\bar{c}_j = c_j - \pi^T A_j$$

only to prove optimality. At other steps, partial pricing is enough (in the primal simplex; it is not possible in the dual simplex).

(2) Columns A_j of non-basic variables come from the initial tableau

this is often sparse (in particular with combinatorial problems, e.g. there are only 2 entries ≠ 0 in a vertex-edge-incidence matrix)

=> can exploit techniques to save memory usage and runtime (data structures and algorithms for sparse matrices)

(3) Maintain $CARRY^{(\ell)}$ implicitly

since $CARRY^{(\ell)}$ is obtained in a simple way from $CARRY^{(\ell-1)}$, it is not necessary to store the complete matrix $CARRY^{(\ell)}$.

$CARRY^{(\ell)} = P_\ell \cdot CARRY^{(\ell-1)}$

with

$P_\ell = (e_1, \ldots, e_{r-1}, \eta, e_{r+1}, \ldots, e_m)$ models elementary row operations

e_i = i-th unit vector

$$\eta = \begin{pmatrix} -\frac{x_{1s}}{x_{rs}} \\ \vdots \\ \frac{1}{x_{rs}} \\ \vdots \\ -\frac{x_{ms}}{x_{rs}} \end{pmatrix} \leftarrow \text{pivot row}$$

=> store P_ℓ by its η-vector and position r

inductively

$$CARRY^{(\ell)} = P_{\ell-1} \cdot P_{\ell-2} \cdot \ldots \cdot P_1 \cdot CARRY^{(0)}$$

if ℓ gets large, one can "re-invert" , i.e., the search for an equivalent but shorter sequence of η-vectors ($\ell \leq m$ suffice)

(4) Combine these techniques with methods for numerical stability

LU-partition, Cholesky-factorization

(Class on numerical methods)

Every regular matrix B can be written as $B = P \cdot L \cdot U$ with

P = permutation matrix (then $P^{-1} = P^T$)

L = lower triangular matrix

U = upper triangular matrix

Linear systems $Bx = b$ can then be easily solved:

$Bx = b$ transforms to $LUx = P^T b$

solve $Ly = P^T b$

solve $Ux = y$

i.e., solve 2 linear systems in triangular form

Some quotations by Bob Bixby, the "father" of Cplex and Gurobi

Bixby
ISMP 2003

from Solving Real-World Linear Programs: A Decade and More of Progress, Operations Research (50) 2002, 3-15

It was thus around 1987 that I became seriously involved in the computational aspects of linear programming. The first version of CPLEX, CPLEX 1.0, was released in 1988.

Advances in computing machinery

Table 1: **Machine improvements–Simplex algorithms**

Old machine/processor	New machine/processor	Estimated speedup
Sun 3/50	Compaq Server ES40, 667 MHz	900
Sun 3/50	Pentium 4, 1.7 GHz	800
25 MHz Intel 386	Compaq Server ES40, 667 MHz	400
IBM 3090/108S	Compaq Server ES40, 667 MHz	45
Cray X-MP/416	Compaq Server ES40, 667 MHz	10

Table 2: **Machine improvements–Barrier algorithms**

Old machine/processor	New machine/processor	Estimated speedup
Sun 3/50	Pentium 4, 1.7 GHz	13000
Sun 3/50	Compaq Server ES40, 667 MHz	12000
33 MHz Intel 386	Compaq Server ES40, 667 MHz	4000
IBM 3090/108S	Compaq Server ES40, 667 MHz	10
Cray X-MP/416	Compaq Server ES40, 667 MHz	5

Algorithmic improvements

The dual simplex algorithm with steepest edge.

The dual simplex algorithm was introduced by Lemke [1954]. It is not a new algorithm. However, to my knowledge, commercial implementations of this algorithm were not available in 1987 as full-fledged alternatives to the primal simplex algorithm. [...]

All that has changed. The dual simplex algorithm is now a standard alternative in modern codes. Indeed, computational tests, some of which will be presented later in this paper, indicate that the overall performance of the dual algorithm may be superior to that of the primal algorithm.

There are a number of reasons why implementations of the dual simplex algorithm have become so powerful. The most important is an idea introduced by Goldfarb and Forrest [1992], a so-called "steepest-edge" rule for selecting the "leaving variable" at each dual simplex iteration. This method requires relatively little additional computational effort per iteration and is far superior to "standard" dual methods, in which the selection of the leaving variable is based only upon selecting a basic variable with large primal infeasibility.

Linear algebra

Linear algebra improvements touch all the parts of simplex algorithms and are also crucial to good implementations of barrier algorithms. Enumerating all such improvements is beyond the scope of this paper. I will mention only a few. For simplex algorithms, two improvements stand out among the rest.

The first of these to be introduced was dynamic LU-factorization using Markowitz threshold pivoting. This approach was perfected by Suhl and Suhl [1990], and has become a standard part of modern codes. In previous-generation codes, "preassigned pivot" sequences were used in the numerical factorization (see

Hellerman and Rarick [1971]). These methods were very effective when no numerical difficulties occurred, but encountered serious difficulties in the alternative case.

The second major linear algebra improvement is that LP codes now take advantage of certain ideas for solving large, sparse linear systems, ideas that have been known in the linear-algebra community for several years (see Gilbert and Peierls [1988]). At each major iteration of a simplex algorithm, several size-able linear systems must be solved. The order of these systems is equal to the number of constraints in the given LP. Typically these systems take as input a vector with a very small number of nonzero entries, say between one and ten – independent of overall model size – and output a vector with only a few additional nonzeros. Since it is unlikely that the sparsity of the output is due to cancellation during the solve, it follows that only a small number of nonzeros in the LU-factorization (and update) of the basis could have been touched during the solve. The trick then is to carry out the solve so that the work is linear in this number of entries, and hence, in total, essentially a constant time operation, even as problem size grows. The effect on large linear programs can be enormous.

Presolve

This idea is made up of a set of problem reductions: Removal of redundant constraints, fixed variables, and other extraneous model elements. The seminal reference on this subject is Brearley et al [1975]. Presolve

was available in MPS III, but modern implementations include a much more extensive set of reductions, including so-called aggregation (substituting out variables, such as free variables, the satisfaction of the bounds of which are guaranteed by the satisfaction of the bounds on the variables that remain in the model). The effects on problem size can be very significant, in some cases yielding reductions by factors exceeding an order of magnitude. Modern presolve implementations are seamless in the sense that problem input and solution output occur in terms of the original model.

Examples of performance improvements

Table 10: **Solution times–Best simplex**

Model	CPLEX 1.0	CPLEX 2.2	CPLEX 5.0	CPLEX 7.1	Algorithm
car	1555.0	701.1	275.8	120.6	primal
continent	364.7	110.5	104.4	46.7	primal
energy1	1217.4	275.0	260.5	22.6	dual
energy2	10130.1	736.0	664.0	693.9	dual
energy3	21797.1	271.9	229.1	161.7	dual
fuel	5619.5	1123.2	698.6	675.0	primal
initial	3832.2	102.2	51.3	15.5	dual
schedule	152404.0	252.3	220.8	64.6	dual

full paper

SOLVING REAL-WORLD LINEAR PROGRAMS: A DECADE AND MORE OF PROGRESS

ROBERT E. BIXBY

ILOG, Inc. and Rice University, bixby@ilog.com or bixby@rice.edu

This paper is an invited contribution to the 50th anniversary issue of the journal *Operations Research*, published by the Institute of Operations Research and Management Science (INFORMS). It describes one person's perspective on the development of computational tools for linear programming. The paper begins with a short personal history, followed by historical remarks covering the some 40 years of linear-programming developments that predate my own involvement in this subject. It concludes with a more detailed look at the evolution of computational linear programming since 1987.

1. INTRODUCTION

I am a relative newcomer to computation. For the first half of my scientific career, my research focused exclusively on the theoretical aspects of operations research and discrete mathematics. That focus began to change in the early 1980s with the appearance of personal computers.

My first PC was used primarily to implement elementary algorithms used in teaching. At first these algorithms did not include a simplex algorithm; eventually, however, I concluded that it would be useful to incorporate computation in the LP courses that I was teaching. As a result, I started writing my own code, initially a simple tableau code.

At that time, in the early 1980s, I knew nothing about the computational aspects of linear programming (LP). I knew a great deal of theory, but numerical analysis and the computational issues associated with numerical algorithms were not subjects that were part of my graduate education. I had no idea that tableaus were numerically unstable.

Fortunately for me, by the time my interests in computation to this day, I'm not quite sure why Tom thought my code would eventually be reasonably good. Initially it certainly was not.

After the code was delivered to Chesapeake, there followed a period of about two years during which I received a steady stream of practical LPs from Chesapeake, LPs on which my code did not do very well. In each case, I poked around in my code and the LP itself to see what ideas I could come up with, never looking in the literature (this wasn't my area of research). Slowly the code got better, until some time around 1986, one of Tom's colleagues informed me that my code had actually gotten good enough that one of their customers was interested in obtaining it separately. I was, to say the least, surprised, and immediately set about doing my first actual comparisons to other LP codes. I chose Roy Marsten's (1981) quite successful and portable (that was key for me) XMP code. I discovered, to my amazement, that for a substantial subset of the *netlib*[1] testset my code was indeed pretty good, running on average two times faster than XMP. In addition, it appeared that my code was significantly more stable than XMP.

This comparison to XMP was an important part of

tation had started, the Department of Industrial Engineering and Management Sciences at Northwestern University had hired Bob Fourer, one of the creators of the AMPL modeling language. Bob had worked for several years at the National Bureau of Economic Research doing practical linear programming, followed by a graduate career at Stanford. He knew a lot about the computational aspects of mathematical programming, and he passed on a great deal of that knowledge to me in informal conversations.

Linear programming become more central to what I was doing when a friend of mine, Tom Baker, founded Chesapeake Decision Sciences (now a part of Aspen Technologies). Shortly thereafter, Tom asked if I had an LP code that he could use in the LP module of the product he was building. I said yes, converted my code to C (that was one of Tom's conditions), and delivered it to him.

This comparison to XMP was an important part of what transformed LP computation into a serious part of my scientific research. Equally important was integer programming.

This was the mid-1980s, and integer-programming computational research was beginning to flower, with important contributions by people such as Martin Grötschel, Ellis Johnson, Manfred Padberg, and Laurence Wolsey. Linear programming was an essential component in that work, but the tools available at that time were proving to be inadequate. The then state-of-the-art codes, such as MPSX/370, simply were not built for this kind of application; in addition, they did not deal well with issues such as degeneracy. The situation at the time is well described by some remarks of Grötschel and Holland (1991), commenting on their use of MPSX/370 in work on the traveling salesman problem: They note that if the LP-package they were using had been

Subject classification: Professional: comments on
Area of review: Anniversary Issue (Special).

0030-364X/02/5001-0003 $05.00
1526-5463 electronic ISSN

Operations Research © 2002 INFORMS
Vol. 50, No. 1, January–February 2002, pp. 3–15

bixby-or2002.pdf

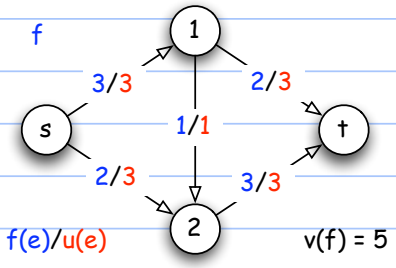
- Goals of this chapter
  - Illustrate the revised simplex algorithm
  - Illustrate how to handle LPs with (exponentially) many columns: Column Generation
  - We use a path-based formulation of the max-flow problem

- The max-flow problem (see ADM I)
  - Maximum Flow problem (MFP)
    - Instance
      - network (G, u, s, t) where
        - G is a digraph
        - s, t are vertices of G, called the source and the sink, respectively
        - $u(e) \geq 0$ is the capacity of edge e
    - Task
      - Find an s,t-flow f with maximum flow value $v(f)$
        - s,t-flow f = edge weight $f(e)$ for every edge with
          - $0 \leq f(e) \leq u(e)$ for all edges e

flow conservation in all vertices $v \neq s, t$

flow value $v(f)$ = net outflow out of s

Example: a flow f



A formulation of the max-flow problem as LP with edge-variables (edge-based formulation)

we use the vertex-edge-incidence matrix as for the shortest path problem

max v (maximize the flow value v) such that

$$Af = b \text{ with } b = \begin{pmatrix} +v \\ -v \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

row s

row t (redundant)

flow conservation

$f \leq u$

$f \geq 0$

Here $E(G) = \{ e_1, \ldots, e_m \}$ and $f = (f_1, \ldots, f_m)^T$, i.e.,, we have a variable per edge for the flow on that edge

A formulation of the max-flow problem as LP with path-variables (path-based formulation)

is based on the Flow Decomposition Theorem of ADM I

ADM I, Theorem 5.2 (Flow Decomposition Theorem for s,t-flows, Ford-Fulkerson 1962)

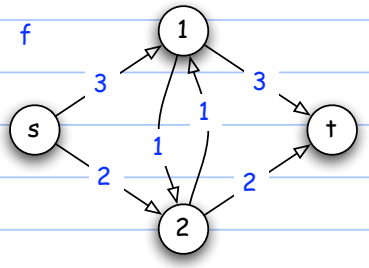
Let $f \neq 0$ be an s,t-flow in (G, u, s, t). Then

f is a positive linear combination of (incidence vectors of) directed (elementary) s,t-paths and directed (elementary) cycles

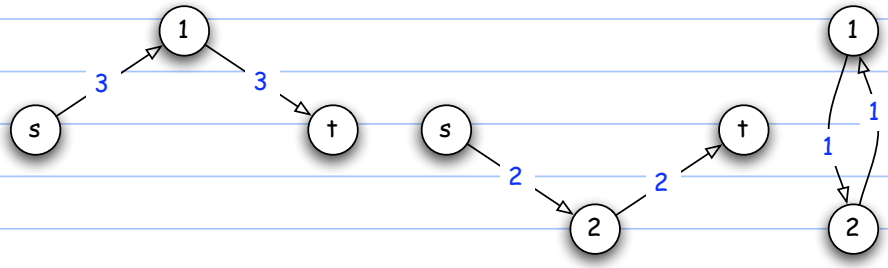
the number of these paths and cycles can is at most m

if f is integer, then there is such a linear combination with integer coefficients

- Example
  - s,t-flow
    - f



  - decomposition into directed paths and cycles (not unique)



  - corresponding linear combination

$$\begin{bmatrix} f_{s1} \\ f_{s2} \\ f_{12} \\ f_{1t} \\ f_{21} \\ f_{2t} \end{bmatrix} \quad f = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 3 \\ 1 \\ 2 \end{bmatrix} = 3 \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + 2 \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} + 1 \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

  - Observe: the s,t-paths determine the flow value, cycles play no role
- The path-based LP
  - Let $P_1, \ldots, P_p$ be all directed elementary s,t-paths in G (Observe: p may be exponential in n).

    The edge-path-incidence matrix $D = (d_{ij})$ is defined by

$$d_{ij} := \begin{cases} 1 & \text{edge } e_i \text{ lies on path } P_j \quad i = 1, \ldots, m \\ 0 & \text{otherwise} \qquad\qquad\qquad\quad j = 1, \ldots, p \end{cases}$$

    $f = (f_1, \ldots, f_p)^T$ is a flow vector that has an entry f_j for s,t-path P_j denoting the amount of flow that is sent along path P_j
  - The capacity constraints read

$Df \le u$

i.e.

$$(d_{i1}, \ldots, d_{ip}) \cdot \begin{pmatrix} f_1 \\ \vdots \\ f_p \end{pmatrix} \le u_i$$

$\sum$ of flows on all paths containing e_i
= flow in edge e_i

for every row e_i

Flow conservation holds trivially and the flow value v is obtained as

$v = \sum_j f_j$

Then the path-based LP is

$\min c^T f$ with $c_j = -1$

$Df \le u$

$f \ge 0$

We transform it into standard form by adding slack variables s_i

$\Rightarrow$ $\min c'^T f'$ with $f' = (f \mid s)^T$, $c' = (c \mid 0)^T$

$D' f' = u$ with $D' = (D \mid I)$ (5.1)

$f' \ge 0$

Slack variable s_i represents the residual capacity on edge e_i

Solving the path-based formulation with the revised simplex algorithm

a basic feasible solution is given by $f = 0$ and $s = u$

$\Rightarrow$ Phase I is not necessary

column D_j of path P_j has (Pricing Rule) reduced cost $\bar{c}_j < 0$

$\Leftrightarrow$ $\bar{c}_j = c_j - \pi^T D_j < 0$

$\Leftrightarrow$ $(-\pi)^T D_j < 1$ because of $c_j = -1$

Interpretation

$-\pi$ = vector of edge weights $-\pi_i$ of edge e_i

$(-\pi)^T D_j$ = length of the path P_j w.r.t. edge weights $-\pi_i$

Idea

Compute the shortest s,t-path P w.r.t. $-\pi$. Then (5.2)

- P has length ≥ 1 => optimality criterion holds for all columns belonging to paths
- P has length < 1 => have found a column (path) to enter the basis
- So it is not necessary to store all (exponentially many) columns of paths explicitly

In general, the idea

Find a column that violates the optimality criterion at most

is again an LP, whose solution is often much simpler than all generating all columns in the revised simplex algorithm explicitly.

This idea is called Column Generation

For the column of slack variable s_i we obtain

- s_i has negative reduced cost <=> $-\pi_i < 0$ (5.3)
- since:
  - reduced cost of column s_i is $0 - (\pi^T I)_i = -\pi_i < 0$

5.1 Theorem (Solving the path-based max-flow problem with column generation)

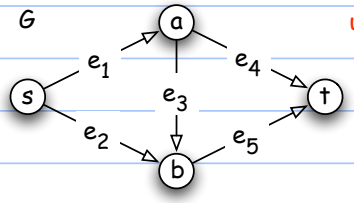
- A basic feasible solution of (5.1) fulfills the optimality criterion
  <=> $-\pi \geq 0$ and the shortest s,t-path w.r.t. $-\pi$ has length ≥ 1
- The pivot steps in the revised simplex algorithm are shortest-path computations with edge lengths $-\pi \geq 0$ or letting a slack variable s_i enter the basis if $-\pi_i < 0$
- Proof
  - If $-\pi_i < 0$ for some i, then s_i is because of (5.3) a non-basic variables with negative reduced cost => optimality criterion violated
  - If $-\pi \geq 0$, then pricing reduces because of (5.2) and (5.3) to computing a shortest s,t-path with non-negative edge weights $-\pi$. ❑

Consequence

- The revised simplex algorithm solves the max-flow problem (essentially) as a sequence of shortest path problems
- Need only a (m+1)x(m+1) CARRY matrix in each iteration

5.1 Example

Prepare data for the revised simplex algorithm



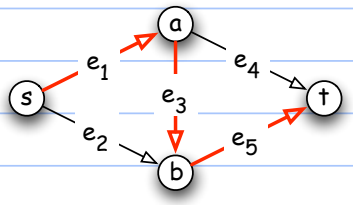
$u_i = 1$

CARRY$^{(0)}$

	0	0	0	0	0	0	← $-\pi^T$
s_1	1	1					
s_2	1		1				
s_3	1			1			
s_4	1				1		
s_5	1					1	

$-\pi^T = 0$ => every s,t-path is a shortest path with cost $0 < 1$

=> choose w.o.l.g. $P_1 = \{ e_1, e_3, e_5 \}$ as entering column



The associated column D_1 of the initial LP is $(1, 0, 1, 0, 1)^T$ and we obtain the transformed column X_1 as

$$X_1 = B^{-1}D_1 = D_1$$

The reduced cost of column X_1 is then

$$c_1 - \pi^T D_1 = -1 + \text{length of the path} = -1 + 0 = -1$$

Data for pivot operation:

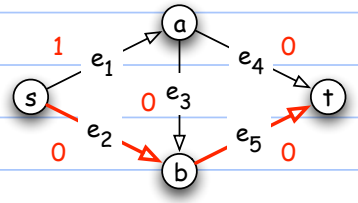
	0	0	0	0	0	0	−1
s_1	1	1					1
s_2	1		1				0
s_3	1			1			1
s_4	1				1		0
s_5	1					1	1

1st pivot

CARRY$^{(1)}$

	1	1	0	0	0	0
f_1	1	1				
s_2	1		1			
s_3	0	-1		1		
s_4	1				1	
s_5	0	-1				1

η-vector



graph with edge weights -π and shortest path

The associated column D_2 of the initial LP is $(0, 1, 0, 0, 1)^T$ and we obtain the transformed column X_2 as

$X_2 = B^{-1}D_2 = D_2$

The reduced cost is c_2 + length shortest path = -1 + 0 = -1

Data for next pivot:

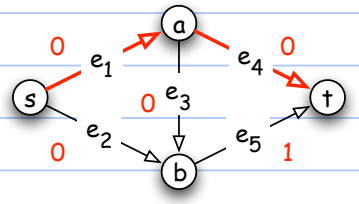
	1	1	0	0	0	0	-1
f_1	1	1					0
s_2	1		1				1
s_3	0	-1		1			0
s_4	1				1		0
s_5	0	-1				1	1

2nd pivot

CARRY$^{(2)}$

	1	0	0	0	0	1
f_1	1	1				
s_2	1	1	1			-1
s_3	0	-1		1		
s_4	1				1	
f_2	0	-1				1

η-vector



graph with edge weights -π and shortest path

The associated column D_3 of the initial LP is $(1, 0, 0, 1, 0)^T$ and we obtain the transformed column X_2 as

$X_3 = B^{-1}D_3 = (1, 1, -1, 1, -1)^T$

The reduced cost is c_3 + length shortest path = -1 + 0 = -1

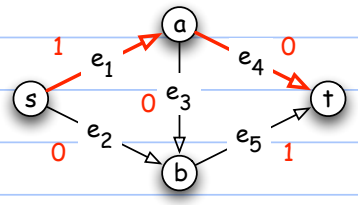
Data for next pivot:

	1	0	0	0	0	1	-1
f_1	1	1					1
s_2	1	1	1			-1	1
s_3	0	-1		1			-1
s_4	1				1		1
f_2	0	-1				1	-1

3rd pivot

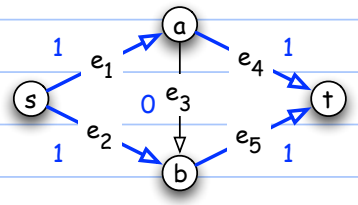
CARRY(3)

	2	1	0	0	0	1
f_3	1	1				
s_2	0	0	1			-1
s_3	1	0		1		
s_4	0	-1			1	
f_2	1	0				1



length shortest path ≥ 1

=> have reached an optimal solution with maximum flow $f_3 + f_2$



maximum flow

Remarks

- Column generation maintains only a (changing) subproblem of the initial problem (called the master problem) with few columns. It generates new columns when needed for improvement by solving the pricing operation as a separate optimization problem over all possible (and only implicitly represented) columns.
- In the max-flow example, the pricing reduces to a simple shortest path problem. In general, the pricing will be more difficult and often even NP-hard (e.g. if the flow carrying paths must respect additional constraints, e.g. in traffic applications where routes must not be too long, or only main streets ...).
- Column generation is one of the work horses for solving complex network problems. More in ADM III

- Goal of this chapter
  - modify the simplex algorithm so that lower and upper bounds can be handled implicitly
  - serves as preparation for the network simplex (programming exercise)
  - lower bounds are easy
  - upper bounds need a modified definition of a basic feasible solution, this then leads to a more efficient and rather easy variation of the simplex algorithm

- LP in standard form with lower and upper bounds
  - min $c^T x$ s.t. $Ax = b$, $\ell \le x \le u$

$$\ell = \begin{pmatrix} \ell_1 \\ \vdots \\ \ell_n \end{pmatrix} \geq 0 \qquad \text{vector of lower bounds}$$

$$u = \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} \geq \ell \qquad \text{vector of upper bounds}$$

- Approaches
  - (1) treat bounds as additional constraints and transform the resulting LP into standard form with slack variables for the bound constraints

    => matrix A gets larger

    unwanted, as bounds are very simple constraints
  - (2) treat bounds implicitly by a slight variation of the simplex algorithm

    -> in this chapter

- Treat lower bounds by substituting variables
  - Approach: $x_j = y_j + \ell_j$

  => $y_j = x_j - \ell_j$

  => write initial LP as

  min $c^T y$ s.t. $Ay = b'$, $0 \le y \le u'$

  with $b' := b - A\ell$ and $u' := u - \ell$

  Solution y of the modified LP yields a solution x of the initial LP by

  $x = y + \ell$

So assume w.o.l.g. $\ell = 0$ in the sequel

i.e., $\min c^T x$

$s.t.\ Ax = b \qquad (5.4)$

$0 \le x \le u$

and w.o.l.g. rank(A) = m

Basic feasible solutions for LPs with upper bounds

Extended partition of the variables

Instead of a partition of the variables/columns of A into B (basic variables) and N (non-basic variables) we now consider an extended partition into

B basic variables

L non-basic variables with value = lower bound = 0

U non-basic variables with value = upper bound = u_j

for such a partition we want that

$$Bx_B + Lx_L + Ux_U = Bx_B + Uu_U = b \qquad (5.5)$$

which gives

$$x_B = B^{-1}(b - Uu_U) = B^{-1}b - B^{-1}Uu_U$$

A basic feasible solution with basis B of an LP with upper bounds is every basic solution of the form (5.5)

5.2 Theorem (Fundamental Theorem for LPs with upper bounds)

If LP (5.4) has an optimal solution, then also an optimal solution that is a basic feasible solution of the form (5.5).

Proof

Model $x \le u$ with slack variables s as linear system $x + s = u$

Then we obtain the following large LP in standard form (LPSU)

$\min\ c^T x$

$s.t.\quad Ax = b$

$x + s = u$

$x \ge 0,\ s \ge 0$

Since A has full row rank, also (LPSU) has full row rank.

Let $(x,s)^T$ be a basic feasible solution of (LPSU) with basis B'.

Set

$$B := \{ j \in \{1,\dots,n\} \mid x_j \in B' \text{ and } s_j \in B' \} \text{ and } p := |B|$$

$L := \{ j \in \{1,\dots,n\} \mid x_j \notin B' \text{ and } s_j \in B' \}$ and $s := |L|$

$U := \{ j \in \{1,\dots,n\} \mid x_j \in B' \text{ and } s_j \notin B' \}$ and $q := |U|$

Identifying $B = A_B$ and $U = A_U$ we can permute B' such that:

$$PB'Q^T = \begin{pmatrix} B & B_q & 0 & 0 \\ I_p & 0 & I_p & 0 \\ 0 & I_q & 0 & 0 \\ 0 & 0 & 0 & I_s \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \end{matrix} \quad \text{rows for } x + s = u$$

with permutation matrices P and Q

Counting the rows of B' gives $m + n = m + p + q + s$

Counting the columns of B' gibes $m + n = 2p + q + s$

=> $p = m$ => B is an $m \times m$-matrix

Laplace expansion of $\det(B') = \det(PB'Q)$ along the last n rows gives $|\det(B')| = |\det(B)|$

B' is a basis of the large LP => $\det(B') \neq 0$ => $\det(B) \neq 0$

=> B is basis of A

Since $B'(x,s)^T = (b,u)^T$, the permuted form of B' transforms into (5.6)

$$Bx_B + Ux_U = b$$

$$x_B + s_B = u_B$$
$$x_U = u_U$$
$$s_L = u_L$$

So the partition B, L, U defines a basic solution $Bx_B + Ux_U = b$ of the form (5.5)

=> every basic feasible solution of the large LP corresponds to a basic feasible solution of (5.4) in the form of (5.5)

=> statement with the Fundamental Theorem of linear optimization (Theorem 3.12) ❑

Optimality criterion for LPs with upper bounds

5.3 Theorem (Optimality criterion for LPs with upper bounds)

A basic feasible solution of the form (5.5) is optimal

<=> the reduced costs in the tableau fulfill

$$\left.\begin{array}{lll} \bar{c}_j = 0 & \text{for } x_j \in B \\ \bar{c}_j \geq 0 & \text{for } x_j \in L \\ \bar{c}_j \leq 0 & \text{for } x_j \in U \end{array}\right\} \quad (5.7)$$

Proof:

- By transforming the optimality conditions of the large LP ❑

- Choosing the pivot element for LPs with upper bounds
  - Choosing the pivot column
    - Choose column A_j with $j \in L$ and $\bar{c}_j < 0$ or column A_j with $j \in U$ and $\bar{c}_j > 0$ (5.8)
  - Choosing the pivot row
    - $x_s \in L$ => increase x_s as much as possible

      $x_s \in U$ => decrease x_s as much as possible

      Use the representation of $x(\theta)$ according to (3.16)

$$x_\ell(\theta) = \begin{cases} y_{B(i)} - \theta x_{is} & \ell = B(i),\ i = 1,\dots,m \\ \theta & \ell = s \\ 0 & \text{sonst} \end{cases}$$

      and take into account that $x_s = u_s$ is possible
    - $x_{is} < 0$ => $x_{B(i)}$ increases

$$\Rightarrow \text{ bound } \theta_i = \frac{u_i - x_{i0}}{-x_{is}} \text{ for } \theta$$

    - $x_{is} > 0$ => $x_{B(i)}$ decreases

$$\Rightarrow \text{ bound } \theta_i = \frac{x_{i0}}{x_{is}} \text{ for } \theta$$

      (as in the ordinary simplex algorithm)
  - Choose θ as minimum of the θ_i and u_s. There are 2 cases
    - The minimum is attained at u_s
      - leave the basis unchanged, variable x_s moves from U to L or vice versa
    - The minimum is attained at $\theta = \theta_r$
      - pivot with x_{rs}

        the new value of x_s is θ (if $x_s = 0$) or $u_s - \theta$ (if $x_s = u_s$)
- Termination
  - requires additional arguments

- A more detailed treatment of the network simplex algorithm will be done in the exercises

  Here: a proof that it is a specialization of the simplex algorithm with upper bounds

- Network problem (Input for the network simplex)

  min $c^T x$ s.t. $Ax = b$, $0 \le x \le u$

  A = vertex-edge-incidence matrix of a digraph $G = (V, E)$

  In the terminology of ADM I this is a Minimum Cost Flow Problem (MCFP)

- Basic solutions of the network problem
  - we assume in the sequel that $V = \{1,\dots,n\}$ and that G is connected.

    rows of A add up to 0

    => delete w.o.l.g. the row for vertex 1 and denote the resulting matrix again by A

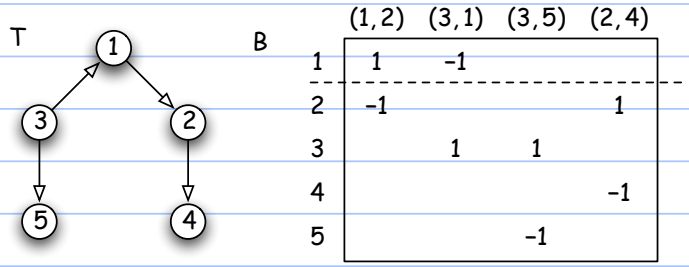
    => A has $n-1$ rows

  - Consider a spanning tree T of G

    => T has $n-1$ edges

Let B be the set of the associated columns of A

Example:



Consider T as undirected tree with root 1

Order the vertices of T in preorder traversal (i.e., root-left-right recursively)

in the example: 1, 3, 5, 2, 4

Order the edges according to this vertex order,

i.e., for each vertex $j \neq 1$ take the (unique) last edge on the path from 1 to j.

in the example: (3,1) (3,5) (1,2) (2,4)

consider the permuted matrix B' according to these row and column orders

	(3,1)	(3,5)	(1,2)	(2,4)
1	-1		1	
3	1	1		
5		-1		
2			-1	1
4				-1

5.4 Lemma

The permuted matrix B´ obtained from the preorder traversal of the tree is (after deleting row 1) an upper triangular matrix with entries ≠ 0 on the diagonal.

Proof

Suppose that the preorder traversal just visits vertex i.

Let j be the father of i in T.

=> (i,j) or (j,i) is tree edge, say (i,j)

Permutation of the rows and columns => (i, (i,j)) is an entry on the diagonal of B´

Preorder traversal => j was visited from i

=> column (i,j) contains

-1 in row j

+1 in row i

0 in all later rows ❑

5.5 Consequence

The rows and columns of the vertex-edge-incidence matrix of a spanning trees of G can be permuted by a preorder traversal so that the resulting matrix is non-singular and upper triangular.

The linear systems $Bx = b$ and $\pi^T B = c_B^T$ of the revised simplex algorithm can easily be solved by exploiting the upper triangular form and the fact that every column contains at most 2 entries ≠ 0. This amounts to simple iterative substitution along the triangular form.

5.6 Theorem (correspondence basis <-> spanning tree)

Every spanning tree of G defines a basis of the network problem (which need not be feasible w.r.t. $0 \le x \le u$).

Every basis of the network problem defines a spanning tree of G.

Proof

=>:

Lemma 5.4.

In particular, every basis has n-1 columns.

<=:

Let B be a basis of the network problem

=> the associated columns correspond to a subgraph G´ with n-1 edges

Claim: G´ has no undirected cycles

Assume that G´ has an undirected cycle C.

choose an orientation of C and let C^+ be the set of forward edges and C^- be the set of backward edges of C w.r.t. the chosen orientation.

every vertex i in C is incident to exactly 2 edges from C

$$\Rightarrow \quad \sum_{e \in C^+} A_e - \sum_{e \in C^-} A_e = 0$$

this contradicts the fact that B is a basis.

ADM I, Theorem 2.3 => every undirected graph with n vertices and n-1 edges and without cycles is a tree ❑

Steps in the revised simplex algorithm

Basis B = tree together with partition B, L, U.

1. Computing the right hand side

It is the solution of

$$Bx_B = b - Uu_U =: b'$$

=> solve the linear system $Bx_B = b'$

iteratively along the triangular form of B according to Consequence 5.4

2. Computing the simplex multipliers π_i

They are the solution of

$$\pi^T B = c_B^T$$

=> $\pi_i - \pi_j = c_{ij}$ for column/edge $(i,j) \in B$

=> iteratively along the triangular form of B , π_i of the last row can be read of directly, then iterate backwards

3. Optimality criterion

- reduced cost of column/edge (i,j) is

  $\bar{c}_{ij} = c_{ij} - \pi^T A_{ij} = c_{ij} - \pi_i + \pi_j$

  Section 5.4 =>

$$\bar{c}_{ij} \geq 0 \quad \text{for } (i,j) \in L$$
$$\bar{c}_{ij} \leq 0 \quad \text{for } (i,j) \in U$$

- 4. Computing the transformed column X_{rs}
  - $-X_{rs}$ corresponds to the change of the basic variables when the value x_{rs} of the non-basic pivot column is set to 1

  so (r,s) enters the basis, B is a tree

  => B + (r,s) contains a unique cycle C

  Orienting C according to (r,s) partitions C into a set of forward edges and a set of backward edges.

  Setting x_{rs} to 1 then implies in C a change by

  +1 on forward edges

  -1 on backward edges

  => pivot operation corresponds exchanging edges on this cycle.

If the non-basic variable x_{rs} is the "bottleneck", then there it changes only from L to U or vice versa (see Section 5.4).

- 5. Computing an initial basic feasible solution
  - = Phase I, see Exercises

- 6. Anti-cycling
  - by considering only strong tree solutions, see Exercises.

- Literature on the network simplex algorithm
  - Chapter 11 in

    K. Ahuja, T.L. Magnanti, J.B. Orlin

    Network Flows: Theory, Algorithms, and Applications

    Prentice Hall, 1993