Supporting hyperplanes

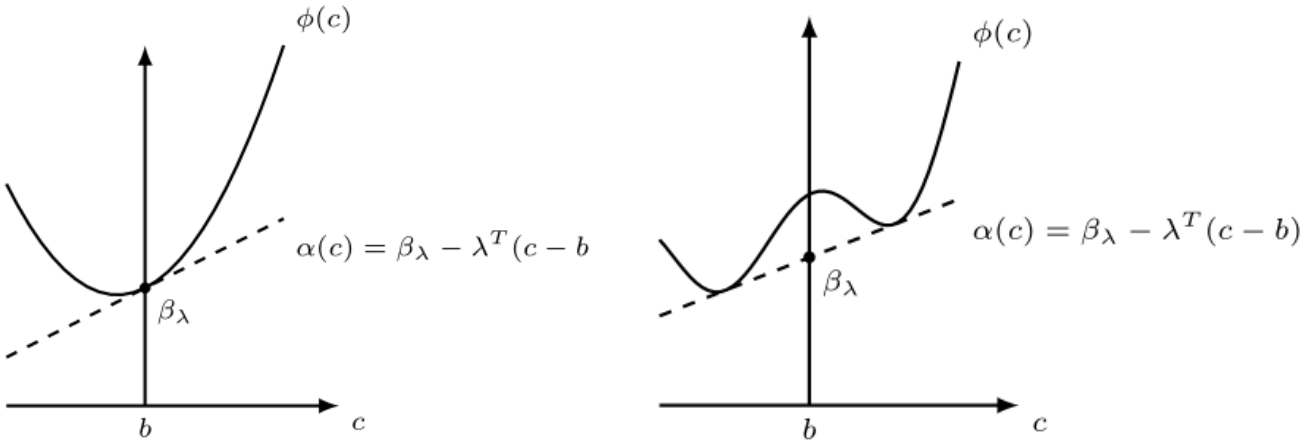


Figure 1: Geometric interpretation of the dual with optimal value $g(\lambda) = \beta_\lambda$. In the situation on the left strong duality holds, and $\beta_\lambda = \phi(b)$. In the situation on the right, strong duality does not hold, and $\beta_\lambda < \phi(b)$.

General approach when using Lagrangian methods

More generally, to

$$\text{minimize } f(x) \text{ subject to } h(x) \leq b,\ x \in X, \tag{1.3}$$

we proceed as follows:

1. Introduce a vector z of slack variables to obtain the equivalent problem

$$\text{minimize } f(x) \text{ subject to } h(x) + z = b,\ x \in X,\ z \geq 0.$$

2. Compute the Lagrangian $L(x, z, \lambda) = f(x) - \lambda^T(h(x) + z - b)$.
3. Define the set

$$Y = \{\lambda \in \mathbb{R}^m : \inf_{x \in X, z \geq 0} L(x, z, \lambda) > -\infty\}.$$

4. For each $\lambda \in Y$, minimize $L(x, z, \lambda)$ subject only to the regional constraints, i.e. find $x^*(\lambda), z^*(\lambda)$ satisfying

$$L(x^*(\lambda), z^*(\lambda), \lambda) = \inf_{x \in X, z \geq 0} L(x, z, \lambda). \tag{1.4}$$

5. Find $\lambda^* \in Y$ so that $(x^*(\lambda^*), z^*(\lambda^*))$ is feasible, i.e. so $x^*(\lambda^*) \in X$, $z^*(\lambda^*) \geq 0$, and $h(x^*(\lambda^*)) + z^*(\lambda^*) = b$. By Theorem 1.2, $x^*(\lambda^*)$ is optimal for (1.3).

Lecture 1 homework

Homework

1. Consider P_b when $f(x) = -x$, and in two cases
   (a) $h(x) = \sqrt{x}$, $b = 2$;
   (b) $h(x) = x^2$, $b = 16$.

   What happens when you try to solve these problems using Lagrangian methods? In each case, find $\phi(b)$ and explain whether strong duality holds.

2. Given $a_1, \ldots, a_n > 0$,

$$\begin{array}{ll} \text{minimize} & -\sum_{i=1}^{n} \log(a_i + x_i) \\ \text{subject to} & x_1, \ldots, x_n \geq 0 \text{ and } \sum_i x_i = b. \end{array}$$

   The optimal x corresponds to one that can be found by a so-called 'water filling algorithm'. Imagine placing bars of heights a_i side by side in the fashion of a histogram and then flooding above these bars so as to cover area of b. Draw a picture to illustrate this idea.

Shadow prices

Theorem 2.5. *Suppose that f and h are continuously differentiable on $\mathbb{R}^n$, and that there exist unique functions $x^* : \mathbb{R}^m \to \mathbb{R}^n$ and $\lambda^* : \mathbb{R}^m \to \mathbb{R}^m$ such that for each $b \in \mathbb{R}^m$, $h(x^*(b)) = b$ and $f(x^*(b)) = \phi(b) = \inf\{f(x) - \lambda^*(b)^T(h(x) - b) : x \in \mathbb{R}^n\}$. If x^* and λ^* are continuously differentiable, then*

$$\frac{\partial \phi}{\partial b_i}(b) = \lambda_i^*(b).$$

Proof. We have that

$$\begin{aligned}\phi(b) &= f(x^*(b)) - \lambda^*(b)^T(h(x^*(b)) - b) \\ &= f(x^*(b)) - \lambda^*(b)^T h(x^*(b)) + \lambda^*(b)^T b.\end{aligned}$$

Taking partial derivatives

$$\begin{aligned}\frac{\partial \phi(b)}{\partial b_i} = \sum_{j=1}^{n} \left(\frac{\partial f}{\partial x_j}(x^*(b)) - \lambda^*(b)^T \frac{\partial h}{\partial x_j}(x^*(b)) \right) \frac{\partial x_j^*}{\partial b_i}(b) \\ - \frac{\partial \lambda^*(b)^T}{\partial b_i}(h(x^*(b)) - b) + \lambda^*(b)^T \frac{\partial b}{\partial b_i}.\end{aligned}$$

The first term on the right-hand side is zero, because $L(x, \lambda^*(b))$ is stationary with respect to x_j at $x^*(b)$. The second term is zero as well, because $x^*(b)$ is feasible and thus $(h(x^*(b)) - b) = 0$. The claim follows. □

A linear programming problem

$$\min \{c^T x : Ax \geq b, x \geq 0\}$$

$$\begin{array}{ll} \text{minimize} & -(x_1 + x_2) \\ \text{subject to} & x_1 + 2x_2 \leq 6 \\ & x_1 - x_2 \leq 3 \\ & x_1, x_2 \geq 0 \end{array}$$

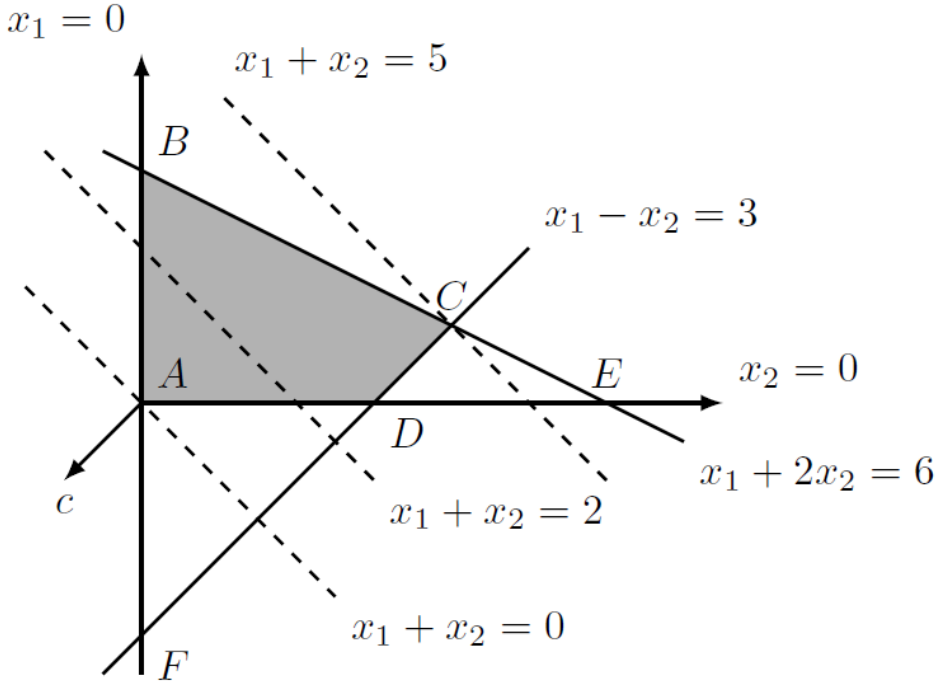


Figure 2: Geometric interpretation of the linear program of Example 2.3

Lecture 3

The simplex algorithm is due to George Dantzig. In his paper on the origins of the simplex algorithm he writes:

"In the summer of 1947, when I first began to work on the simplex method for solving linear programs, the first idea that occurred to me is one that would occur to any trained mathematician, namely the idea of step by step descent (with respect to the objective function) along edges of the convex polyhedral set from one vertex to an adjacent one. I rejected this algorithm outright on intuitive grounds - it had to be inefficient because it proposed to solve the problem by wandering along some path of outside edges until the optimal vertex was reached. I therefore began to look for other methods which gave more promise of being efficient, such as those that went directly through the interior."

This is interesting because Dantzig says that the simplex method is something that any trained mathematician would think of - but that that on first sight it appears to be inefficient.

In practice these days computers solve problems with 100,000s constraints and variables. There is a nice on line simplex method tool which you might like to use to check answers that you obtain by hand when doing questions on Examples sheet 1. There is also a very nice program here, which runs under Windows.

http://trin-hosts.trin.cam.ac.uk/fellows/dpk10/IB/simple2x.html

It was written by Doug Kennedy at Trinity. It takes the hard labour out of performing pivot operations and it may be configured to prompt the choice of pivot elements, or to solve problems automatically.

The simplex tableau

	B (m)	N ($n-m$)	1
m	$A_B^{-1}A_B = I$	$A_B^{-1}A_N$	$A_B^{-1}b$
1	$c_B^T - c_B^T A_B^{-1} A_B = 0$	$c_N^T - c_B^T A_B^{-1} A_N$	$-c_B^T A_B^{-1} b$

$$
\begin{aligned}
\text{minimize} \quad & -(x_1 + x_2) \\
\text{subject to} \quad & x_1 + 2x_2 \leq 6 \\
& x_1 - x_2 \leq 3 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

	x_1	x_2	z_1	z_2	a_{i0}
z_1	1	2	1	0	6
z_2	1	−1	0	1	3
a_{0j}	1	1	0	0	0

	x_1	x_2	z_1	z_2	a_{i0}
z_1	0	3	1	−1	3
x_1	1	−1	0	1	3
a_{0j}	0	2	0	−1	−3

	x_1	x_2	z_1	z_2	a_{i0}
x_2	0	1	$\frac{1}{3}$	$-\frac{1}{3}$	1
x_1	1	0	$\frac{1}{3}$	$\frac{2}{3}$	4
a_{0j}	0	0	$-\frac{2}{3}$	$-\frac{1}{3}$	−5

Simple example with cycling

$$
\begin{aligned}
\text{Max} \quad & I = 2.3x_1 + 2.15x_2 - 13.55x_3 - 0.4x_4, \\
\text{subject to} \quad & 0.4x_1 + 0.2x_2 - 1.4x_3 - 0.2x_4 \leq 0, \\
& -7.8x_1 - 1.4x_2 + 7.8x_3 + 0.4x_4 \leq 0, \\
& x_i \geq 0, \quad i = 1 \ldots 4.
\end{aligned} \tag{1}
$$

x_1	x_2	x_3	x_4	x_5	x_6	I			
0.4	0.2	-1.4	-0.2	1.0			=	0	
-7.8	-1.4	7.8	0.4		1.0		=	0	$T^{(1)}$
-2.3	-2.15	13.55	0.4			1.0	=	0	
1.0	0.5	-3.5	-0.5	2.5			=	0	
	2.5	-19.5	-3.5	19.5	1.0		=	0	$T^{(2)}$
	-1.0	5.5	-0.75	5.75		1.0	=	0	
1.0		0.4	0.2	-1.4	-0.2		=	0	
	1.0	-7.8	-1.4	7.8	0.4		=	0	$T^{(3)}$
		-2.3	-2.15	13.55	0.4	1.0	=	0	

The pivot rule being used is steepest descent: choose the column with the most negative entry in the bottom row.

Beale's example of cycling in linear programming.

We are minimizing. After 6 iterations the tableau is the same as at the start.

The pivot column selection rule is steepest descent: choose the column with the most negative entry in the bottom row.

Note that there is degeneracy, in that there are 0s in the right hand column.

The value of the objective function stays at 0. However, the true minimum is -1/20.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
	$\frac{1}{4}$	-60	$-\frac{1}{25}$	9	1	0	0	0
	$\frac{1}{2}$	-90	$-\frac{1}{50}$	3	0	1	0	0
	0	0	1	0	0	0	1	1
Payoff	$-\frac{3}{4}$	150	$-\frac{1}{50}$	6	0	0	0	0

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_1	1	-240	$-\frac{4}{25}$	36	4	0	0	0
	0	30	$\frac{3}{50}$	-15	-2	1	0	0
	0	0	1	0	0	0	1	1
Payoff	0	-30	$-\frac{7}{50}$	33	3	0	0	0

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_1	1	0	$\frac{8}{25}$	-84	-12	8	0	0
x_2	0	1	$\frac{1}{500}$	$-\frac{1}{2}$	$-\frac{1}{15}$	$\frac{1}{30}$	0	0
	0	0	1	0	0	0	1	1
Payoff	0	0	$-\frac{2}{25}$	18	1	1	0	0

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_3	$\frac{25}{8}$	0	1	$-\frac{525}{2}$	$-\frac{75}{2}$	25	0	0
x_2	$-\frac{1}{160}$	1	0	$\frac{1}{40}$	$\frac{1}{120}$	$-\frac{1}{60}$	0	0
	$-\frac{25}{8}$	0	0	$\frac{525}{2}$	$\frac{75}{2}$	-25	1	1
Payoff	$\frac{1}{4}$	0	0	-3	-2	3	0	0

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_3	$-\frac{125}{2}$	10500	1	0	50	-150	0	0
x_4	$-\frac{1}{4}$	40	0	1	$\frac{1}{3}$	$-\frac{2}{3}$	0	0
	$\frac{125}{2}$	-10500	0	0	-50	150	1	1
Payoff	$-\frac{1}{2}$	120	0	0	-1	1	0	0

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_5	$-\frac{5}{4}$	210	$\frac{1}{50}$	0	1	-3	0	0
x_4	$\frac{1}{6}$	-30	$-\frac{1}{150}$	1	0	$\frac{1}{3}$	0	0
	0	0	1	0	0	0	1	1
Payoff	$-\frac{7}{4}$	330	$\frac{1}{50}$	0	0	-2	0	0

lp_solve.exe

lp_solve.exe [options] "<" <input_file>

```
list of options:
-h              prints this message
-v              verbose mode, gives flow through the program
-d              debug mode, all intermediate results are printed,
                and the branch-and-bound decisions
-p              print the values of the dual variables
-i              print all intermediate valid solutions.
                Can give you useful solutions even if the
                total run time is too long
-t              trace pivot selection
```

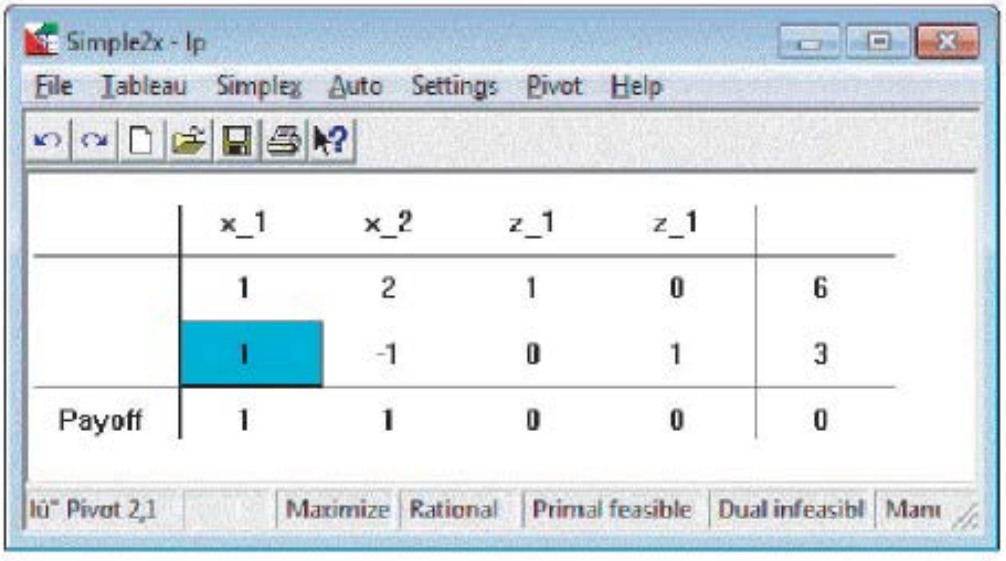
LP0 is a file containing lines of:

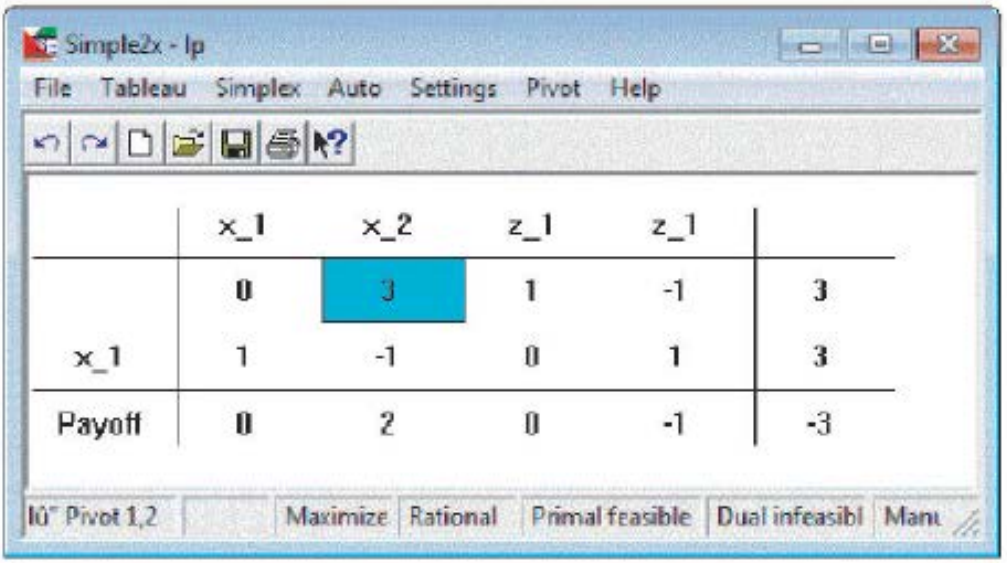
```
          x1 +   x2;
row1: x1 + 2 x2 <= 6;
row2: x1 -   x2 <= 3;
```

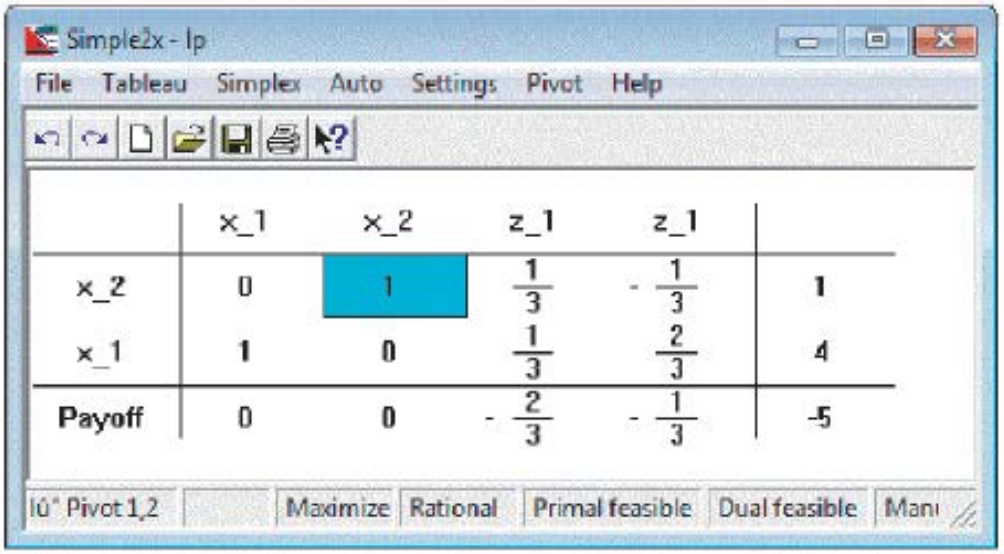
```
>lp_solve -p  < LP0

Value of objective function:  5
x1                            4
x2                            1

Dual values:
row1                  0.66667
row2                  0.33333
```

```
In[127]:= b={6,3}
          c={1,1}
          m={{1,2},{1,-1}};

AbsoluteTiming[
   For[i=1,i<=100000,i++,
   x=LinearProgramming[-c,-m,-b]];
]
Print[x];

Out[128]= {6,3}
Out[129]= {1,1}
Out[130]= {4.0900000,Null}
          {4,1}


The problem is solved in
about 0.000004 seconds.

The solutionis x1=4, x2=1.
```

Simple2x


Simple2x - lp
File Tableau Simplex Auto Settings Pivot Help
x_1 x_2 z_1 z_1
1 2 1 0 6
1 -1 0 1 3
Payoff 1 1 0 0 0
lú" Pivot 2,1
Maximize Rational Primal feasible Dual infeasibl Manı


Simple2x - lp
File Tableau Simplex Auto Settings Pivot Help
x_1 x_2 z_1 z_1
0 3 1 -1 3
x_1 1 -1 0 1 3
Payoff 0 2 0 -1 -3
lú" Pivot 1,2
Maximize Rational Primal feasible Dual infeasibl Manı


Simple2x - lp
File Tableau Simplex Auto Settings Pivot Help
x_1 x_2 z_1 z_1
x_2 0 1 1/3 -1/3 1
x_1 1 0 1/3 2/3 4
Payoff 0 0 -2/3 -1/3 -5
lú" Pivot 1,2
Maximize Rational Primal feasible Dual feasible Manı

CPLEX Optimizer

High-performance mathematical programming solver for linear programming, mixed integer programming, and quadratic programming

IBM ILOG CPLEX Optimization Studio Community Edition

→ Download the Community Edition

Model business issues mathematically and solve them with IBM ILOG CPLEX Optimizer's powerful algorithms to produce precise and logical decisions.

IBM ILOG CPLEX Optimizer's mathematical programming technology enables decision optimization for improving efficiency, reducing costs, and increasing profitability.

- Fundamental algorithms: IBM ILOG CPLEX Optimizer provides flexible, high-performance mathematical programming solvers for linear programming, mixed integer programming, quadratic programming, and quadratically constrained programming problems. These include a distributed parallel algorithm for mixed integer programming to leverage multiple computers to solve difficult problems.

Community Edition

Problem size limited to 1000 variables and 1000 constraints. All features included. Available on the most popular supported platforms.

Two phase method

$$
\begin{array}{lll}
\text{maximize} & -6x_1 - 3x_2 & \\
\text{subject to} & x_1 + x_2 & \geq 1 \\
& 2x_1 - x_2 & \geq 1 \\
& 3x_2 & \leq 2 \\
& x_1, x_2 & \geq 0,
\end{array}
\qquad
\begin{array}{lll}
\text{maximize} & -6x_1 - 3x_2 & \\
\text{subject to} & x_1 + x_2 - z_1 & = 1 \\
& 2x_1 - x_2 - z_2 & = 1 \\
& 3x_2 + z_3 & = 2 \\
& x_1, x_2, z_1, z_2, z_3 & \geq 0.
\end{array}
$$

$$
\begin{array}{lll}
\text{minimize} & y_1 + y_2 & \\
\text{subject to} & x_1 + x_2 - z_1 + y_1 & = 1 \\
& 2x_1 - x_2 - z_2 + y_2 & = 1 \\
& 3x_2 + z_3 & = 2 \\
& x_1, x_2, z_1, z_2, z_3, y_1, y_2 & \geq 0,
\end{array}
$$

	x_1	x_2	z_1	z_2	z_3	y_1	y_2	
y_1	1	1	−1	0	0	1	0	1
y_2	2	−1	0	−1	0	0	1	1
z_3	0	3	0	0	1	0	0	2
II	−6	−3	0	0	0	0	0	0
I	3	0	−1	−1	0	0	0	2

	x_1	x_2	z_1	z_2	z_3	y_1	y_2	
y_1	1	1	−1	0	0	1	0	1
y_2	2	−1	0	−1	0	0	1	1
z_3	0	3	0	0	1	0	0	2
II	−6	−3	0	0	0	0	0	0
I	3	0	−1	−1	0	0	0	2

Phase I now proceeds by pivoting on a_{21} to get

	x_1	x_2	z_1	z_2	z_3	y_1	y_2	
	0	$\frac{3}{2}$	−1	$\frac{1}{2}$	0	1	$-\frac{1}{2}$	$\frac{1}{2}$
	1	$-\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	$\frac{1}{2}$	$\frac{1}{2}$
	0	3	0	0	1	0	0	2
II	0	−6	0	−3	0	0	3	3
I	0	$\frac{3}{2}$	−1	$\frac{1}{2}$	0	0	$-\frac{3}{2}$	$\frac{1}{2}$

and on a_{14} to get

	x_1	x_2	z_1	z_2	z_3	y_1	y_2	
	0	3	−2	1	0	2	−1	1
	1	1	−1	0	0	1	0	1
	0	3	0	0	1	0	0	2
II	0	3	−6	0	0	6	0	6
I	0	0	0	0	0	−1	−1	0

x_1	x_2	z_1	z_2	z_3	
0	3	−2	1	0	1
1	1	−1	0	0	1
0	3	0	0	1	2
0	3	−6	0	0	6

x_1	x_2	z_1	z_2	z_3	
0	1	$-\frac{2}{3}$	$\frac{1}{3}$	0	$\frac{1}{3}$
1	0	$-\frac{1}{3}$	$-\frac{1}{3}$	0	$\frac{2}{3}$
0	0	2	−1	1	1
0	0	−4	−1	0	5

Dual simplex algorithm

$$
\begin{array}{ll}
\text{minimize} & 2x_1 + 3x_2 + 4x_3 \\
\text{subject to} & x_1 + 2x_2 + x_3 - z_1 = 3 \\
& 2x_1 - x_2 - 3x_3 - z_2 = 4 \\
& x_1, x_2, x_3, z_1, z_2 \geq 0.
\end{array}
$$

-1	-2	-1	1	0	-3
-2	1	3	0	1	-4
2	3	4	0	0	0

0	$-\frac{5}{2}$	$-\frac{5}{2}$	1	$-\frac{1}{2}$	-1
1	$-\frac{1}{2}$	$-\frac{3}{2}$	0	$-\frac{1}{2}$	2
0	4	7	0	1	-4

0	1	1	$-\frac{2}{5}$	$\frac{1}{5}$	$\frac{2}{5}$
1	0	-1	$-\frac{1}{5}$	$-\frac{2}{5}$	$\frac{11}{5}$
0	0	3	$\frac{8}{5}$	$\frac{1}{5}$	$-\frac{28}{5}$

Gomory's cutting plane method

0	1	1	$-\frac{2}{5}$	$\frac{1}{5}$	$\frac{2}{5}$
1	0	-1	$-\frac{1}{5}$	$-\frac{2}{5}$	$\frac{11}{5}$
0	0	3	$\frac{8}{5}$	$\frac{1}{5}$	$-\frac{28}{5}$

$$x_2 + x_3 - 1z_1 + 0z_2 \leq x_2 + x_3 - \frac{2}{5}z_1 + \frac{1}{5}z_2 = \frac{2}{5},$$

$$x_2 + x_3 - z_1 \leq 0.$$

0	1	1	$-\frac{2}{5}$	$\frac{1}{5}$	0	$\frac{2}{5}$
1	0	-1	$-\frac{1}{5}$	$-\frac{2}{5}$	0	$\frac{11}{5}$
0	0	0	$-\frac{3}{5}$	$-\frac{1}{5}$	1	$-\frac{2}{5}$
0	0	3	$\frac{8}{5}$	$\frac{1}{5}$	0	$-\frac{28}{5}$

0	1	1	-1	0	1	0
1	0	-1	1	0	-2	3
0	0	0	3	1	-5	2
0	0	3	1	0	1	-6

Markov Chains, Computer Proofs, and Average-Case Analysis of Best Fit Bin Packing

E.G. Coffman, Jr.[1], D. S. Johnson[1], P. W. Shor[1], R. R. Weber[2]

Abstract. Many complex processes can be modeled by (countably) infinite, multi-dimensional Markov chains. Unfortunately, current theoretical techniques for analyzing infinite Markov chains are for the most part limited to three or fewer dimensions. In this paper we propose a computer-aided approach to the analysis of higher-dimensional domains, using several open problems about the average-case behavior of the Best Fit bin packing algorithm as case studies. We show how to use dynamic and linear programming to construct potential functions that, when applied to suitably modified multi-step versions of our original Markov chain, yield drifts that are bounded away from 0. This enables us to completely classify the expected behavior of Best Fit under discrete uniform distributions $U\{J,K\}$ when K is small. (Under $U\{J,K\}$, the allowed item sizes are i/K, $1 \le i \le J$, with all J possibilities equally likely.) In addition, we can answer yes to the long-standing open question of whether there exist distributions of this form for which Best Fit yields linearly-growing waste. The proof of the latter theorem relies on a 24-hour computation, and although its validity does not depend on the linear programming package we used, it does rely on the correctness of our dynamic programming code and of our computer's implementation of the IEEE floating point standard.

Markov Chains, Computer Proofs, and Average-Case Analysis of Best Fit Bin Packing

E.G. Coffman, Jr.[1], D. S. Johnson[1], P. W. Shor[1], R. R. Weber[2]

Many complex processes can be modelled by (countably) infinite, multidimensional Markov chains. Unfortunately, theoretical techniques for analysing infinite Markov chains are for the most part limited to three or fewer dimensions. In this paper we propose a computer-aided approach to the analysis of higher-dimensional domains, using several open problems about the average-case behavior of the Best Fit bin packing algorithm as case studies. We show how to use dynamic and lienar programming to construct potential functions that when applied to suitably modified multi-step versions of our original Markov chain, yield drifts that are bounded away from 0. This enables us to completely classify the expected behavior of Best Fit under discrete uniform distributions U{J, K) when K is small. In addition, we can answer yes to the long-standing open question of whether there exist distributions of this form for which Best Fit yields linearly-growing waste. **The proof of the latter theorem relies on a 24-hour computation, and although its validity does not depend on the linear programming package we used, it does rely on the correctness of our dynamic programming code and of our computer's implementation of the IEEE floating point standard.**

These results are based on the use of a particular kind of potential function (called a *Lyapunov* function in the literature). Once one goes beyond three dimensions, there are no longer simple constructions for the needed functions. There are, however, algorithmic approaches to determining whether Lyapunov functions of certain standard types exist, and the remainder of Section 2 describes them. We show how linear programming can be used to test for the existence of both linear and quadratic Lyapunov functions, using an old lemma of Hajek [9] and a new one of our own.

Maximize γ, subject to

$$\sum_{i=1}^{d} a_i \Delta_i(s) < -\gamma, \textit{ for all } s \in S_k,$$

$$a_i \geq 0, \ 1 \leq i \leq d, \textit{ and}$$

$$\sum_{i=1}^{d} a_i \leq 1.$$

Example: On-line bin packing

An infinite sequence of items are to be packed into bins of size 5. Each successive item is equally likely to be of sizes $1, 2$ or 3. One possible packing algorithm is *Best Fit* (BF), which puts each item into the smallest gap into which it will fit amongst the existing gaps in partially full bins, or if there is no gap large enough, the item goes into an empty bin.

The state at time t is written $x = (x_1, x_2, x_3, x_4)^T$ where x_i is the number of partially full bins with a gap of size i. Then, for example,

$$(0,4,0,0) \longrightarrow \begin{array}{llll} (1,3,0,0) & & 1 & \frac{1}{3} \\ (0,3,0,0) & \text{on arrival of a} & 2 \text{ w.p.} & \frac{1}{3} \\ (0,5,0,0) & & 3 & \frac{1}{3} \end{array}$$

Let $d(x) = E[x(t+1) - x(t) \mid x(t) = x]$, be the *expected drift*, e.g., $d(0,4,0,0) = \left(\frac{1}{3}, -\frac{1}{3}, 0, 0\right)^T$.

Problem: Let $E|x(t)| = x_1 + x_2 + x_3 + x_4$ be the number of partially full bins. Does $E|x(t)| \to \infty$ or is $E|x(t)|$ bounded as $t \to \infty$?

Practical considerations suggest that it is better if $E|x(t)|$ is bounded than if $E|x(t)| \to \infty$.

Theorem *$E|x(t)|$ is bounded as $t \to \infty \iff \exists$ $\delta < 0$ and potential function $\phi(x) \geq 0$ such that*

$$E[\phi(x(t+1)) - \phi(x) \mid x(t) = x] \leq \delta < 0 \text{ for all } x.$$

This theorem makes sense because it says that from every possible starting state the value of the potential function is drifting (on average) towards 0.

Suppose we try $\phi(x) = \sum_{i=1}^{4} a_i x_i$ and hunt for a_i that will work by considering the LP

$$\begin{aligned} \text{minimize} \quad & \delta \\ \text{subject to} \quad & d(x)^T a \leq \delta, \quad \text{for all } x \\ & a_1, a_2, a_3, a_4 \geq 0. \end{aligned}$$

If the optimal solution is $\delta < 0$ this proves that $E|x(t)|$ is bounded as $t \to \infty$.

This is what happens for our example. The LP has only 6 constraints.

For packing items of sizes $1, \ldots, 8$ into bins of size 14 a version of this method gives a LP with 415,953 constraints and takes 24 hours to construct and solve!

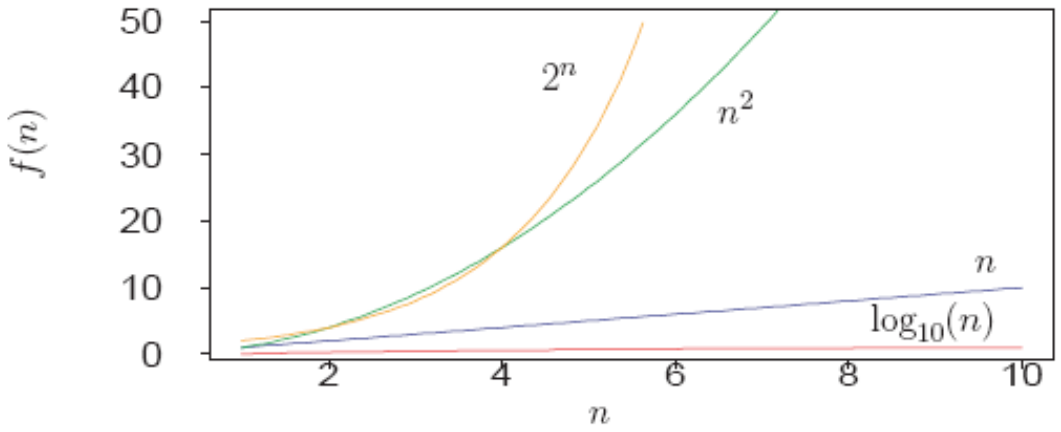
For packing $1, \ldots, 8$ into bins of size 11 the answer can be shown to be $E|x(t)| \to \infty$.

	K	5	6	7	8	9	10	11	12	13	14
J	3	B-L2	B-L2	B-L2	B-Q1	B-Th	B-Th	B-Th	B-Th	B-Th	B-Th
	4		B-L4	B-L3	B-Q1	B-Q1	B-Q1	B-Q1	B-Q1	B-Q1	B-Th
	5			B-L23	B-Q1	B-Q1	B-Q1	B-Q1	B-Q1	B-Q1	B-Q1
	6				B-Q2	B-Q5	B-Q1	B-Q1	B-Q1	B-Q1	B-Q1
	7					B-Q7	B-Q15	B-Q2	B-Q2	B-Q1	B-Q1
	8						B-Q13	Ln-P	B-x	B-x	B-Q7
	9							B-x	Ln-P	Ln-x	B-x
	10								B-x	Ln-x	Ln-x
	11									B-x	Ln-x
	12										B-x

TABLE 2. Results proved for $E[w_{BF}(L_{N,J,K})]$.

Section 3 then uses this technology to prove bounded expected waste for many of the entries in Table 1. A *proof* in this case starts with the construction by computer of a linear program that, although it has few variables, may sometimes have more than 100,000 constraints. The linear program is then solved using a standard LP package. The package we use is CPLEX™ (CPLEX is a trademark of CPLEX Optimization, Inc.), but the validity of our proofs is independent of the correctness of CPLEX, since after CPLEX generates a solution, we verify the validity of that solution using our own code. The correctness of our proofs does, however, depend on the correctness of our generation and checking programs (for which we shall present listings of key routines in the final paper), and on the fact that our computer runs properly and correctly implements the IEEE floating point standard [1].

The running time of algorithms



Suppose instances of size 1000 can be solved in one second.

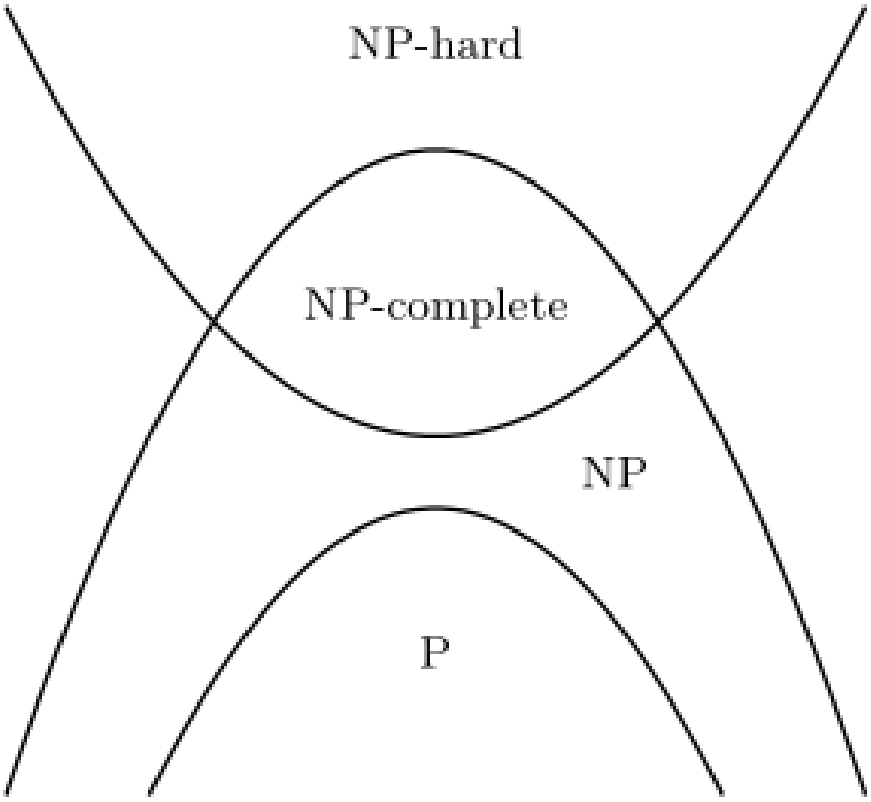
Technology increases computing speed by a factor of 16.

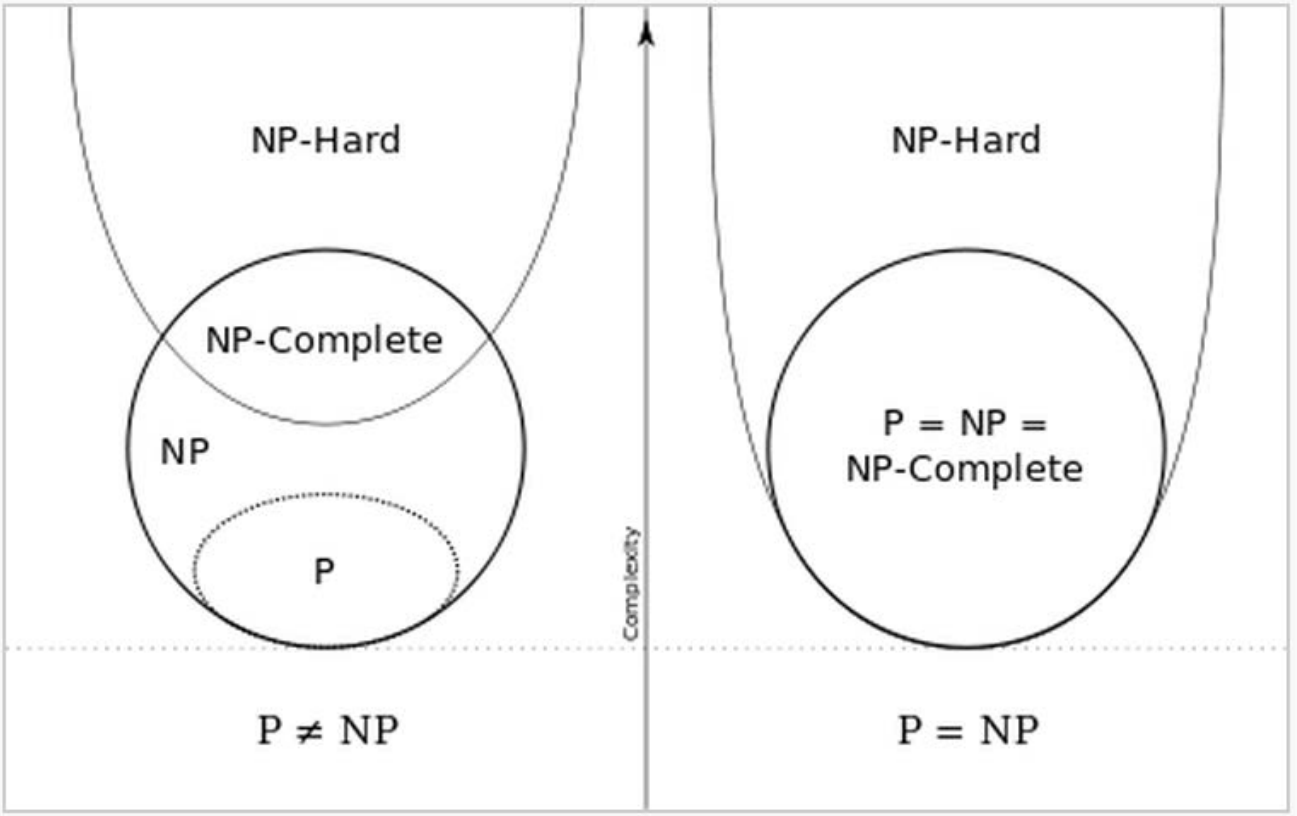
What size of problem can we now solve in one second?

If $T(n) = n^2$ we can now solve problems of size 4000.

If $T(n) = 2^n$ we can now solve problems of size 1004.

Relationship between complexity classes





Euler diagram for P, NP, NP-complete, and NP-hard set of problems

P v NP

Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe! Thus no future civilization could ever hope to build a supercomputer capable of solving the problem by brute force; that is, by checking every possible combination of 100 students. However, this apparent difficulty may only reflect the lack of ingenuity of your programmer. In fact, one of the outstanding problems in computer science is determining whether questions exist whose answer can be quickly checked, but which require an impossibly long time to solve by any direct procedure. Problems like the one listed above certainly seem to be of this kind, but so far no one has managed to prove that any of them really are so hard as they appear, i.e., that there really is no feasible way to generate an answer with the help of a computer. Stephen Cook and Leonid Levin formulated the P (i.e., easy to find) versus NP (i.e., easy to check) problem independently in 1971.

A Turing machine

The machine has states A and B. In each state, the machine reads the bit under the head and executes the instructions in the following table (where Pn prints bit n, L means “move left” and R means “move right, and A and B mean “switch to that state”).

	A	B
0	P1, R, B	P2, L, A
1	P2, L, A	P2, R, B
2	P1, L, A	P0, R, A

					A						
...	0	0	0	0	0	0	0	0	0	0	...
						B					
...	0	0	0	0	1	0	0	0	0	0	...
					A						
...	0	0	0	0	1	2	0	0	0	0	...
				A							
...	0	0	0	0	2	2	0	0	0	0	...

Karp's 21 NP-complete problems

From Wikipedia, the free encyclopedia

- **Satisfiability**: the boolean satisfiability problem for formulas in conjunctive normal form (SAT)
  - **0–1 integer programming** (A variation in which only the restrictions must be satisfied, with no optimization)
  - **Clique** (independent set problem)
    - **Set packing**
    - **Vertex cover**
      - **Set covering**
      - **Feedback node set**
      - **Feedback arc set**
      - **Directed Hamilton circuit**
        - **Undirected Hamilton circuit**
  - **Satisfiability with at most 3 literals per clause** (equivalent to 3-SAT)
    - **Chromatic number** (also called the Graph Coloring Problem)
      - **Clique cover**
      - **Exact cover**
        - **Hitting set**
        - **Steiner tree**
        - **3-dimensional matching**
        - **Knapsack** (Karp's definition of Knapsack is closer to Subset sum)
          - **Job sequencing**
          - **Partition**
            - **Max cut**

https://en.wikipedia.org/wiki/Karp's_21_NP-complete_problems

Firing squad problem

Firing Squad Synchronization, Computer Science's Most Macabre-Sounding Problem

Written by **BEN RICHMOND**

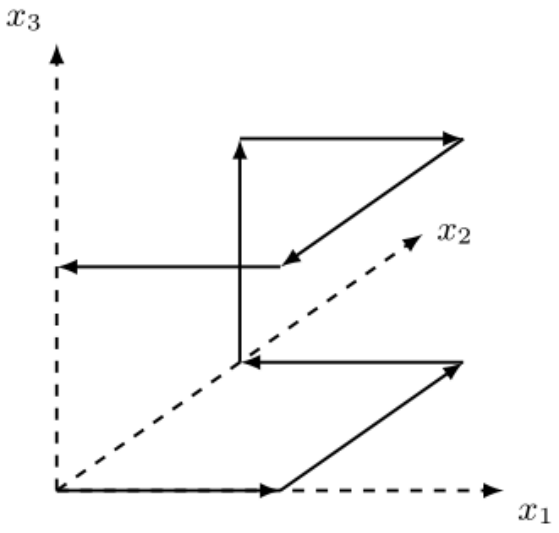
July 14, 2015 // 12:00 PM EST

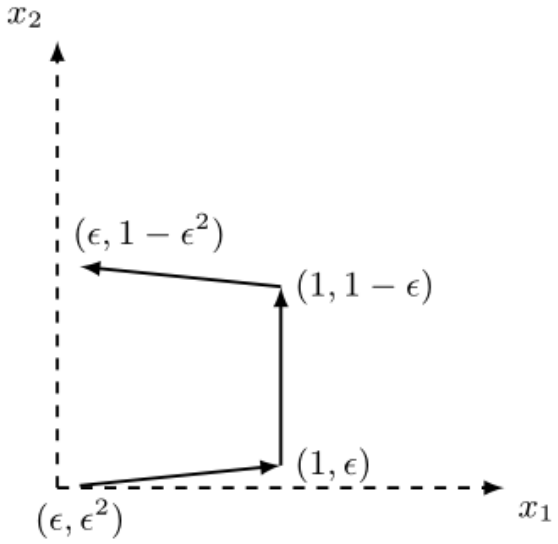
The name of the problem comes from an analogy with real-world firing squads: the goal is to design a system of rules according to which an officer can so command an execution detail to fire that its members fire their rifles simultaneously.

More formally, the problem concerns cellular automata, arrays of finite state machines called "cells" arranged in a line, such that at each time step each machine transitions to a new state as a function of its previous state and the states of its two neighbors in the line. For the firing squad problem, the line consists of a finite number of cells, and the rule according to which each machine transitions to the next state should be the same for all of the cells interior to the line, but the transition functions of the two endpoints of the line are allowed to differ, as these two cells are each missing a neighbor on one of their two sides.

The states of each cell include three distinguished states: "active", "quiescent", and "firing", and the transition function must be such that a cell that is quiescent and whose neighbors are quiescent remains quiescent. Initially, at time t = 0, all states are quiescent except for the cell at the far left (the general), which is active. The goal is to design a set of states and a transition function such that, no matter how long the line of cells is, there exists a time t such that every cell transitions to the firing state at time t, and such that no cell belongs to the firing state prior to time t.

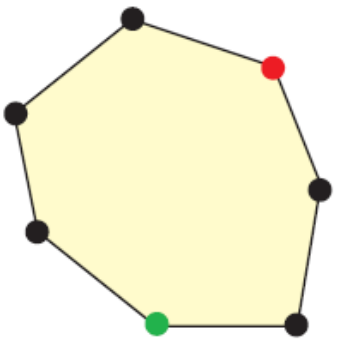
Klee and Minty polytope



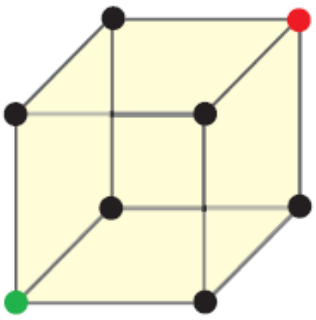


$$\epsilon \leq x_1 \leq 1,$$
$$\epsilon x_{i-1} \leq x_i \leq 1 - \epsilon x_{i-1} \quad \text{for } i = 2, \ldots, n$$

The Hirsch conjecture



$n = 2,\ m = 7$



$n = 3,\ m = 6$

$\Delta(n, m) =$ maximum distance between two vertices of a polytope in $\mathbb{R}^n$ that is defined by m inequalites.

Hirsch Conjecture: $\Delta(n, m) \leq m - n$.

A counterexample to the Hirsch Conjecture was discovered by Francisco Santos in 2010. However, it remains an open question whether or not $\Delta(n, m) =$ is bounded by some polynomial function of m and n.

A counterexample to the Hirsch conjecture

Francisco Santos*

To Victor L. Klee (1925–2007), in memoriam†

Abstract

The Hirsch Conjecture (1957) stated that the graph of a d-dimensional polytope with n facets cannot have (combinatorial) diameter greater than $n-d$. That is, that any two vertices of the polytope can be connected to each other by a path of at most $n-d$ edges.

This paper presents the first counterexample to the conjecture. Our polytope has dimension 43 and 86 facets. It is obtained from a 5-dimensional polytope with 48 facets which violates a certain generalization of the d-step conjecture of Klee and Walkup.

1 Introduction

The Hirsch conjecture is the following very fundamental statement about the combinatorics of polytopes. It was stated by Warren M. Hirsch in 1957 in the context of the *simplex method*, and publicized by G. Dantzig in his 1963 monograph on linear programming [10]:

> The (combinatorial) diameter of a polytope of dimension d with n facets cannot be greater than $n-d$.

Here we call *combinatorial diameter* of a polytope the maximum number of steps needed to go from one vertex to another, where a step consists in traversing an edge. Since we never refer to any other diameter in this paper, we will often omit the word "combinatorial".

Our main result is the construction of a 43-dimensional polytope with 86 facets and diameter (at least) 44. Via products and glueing copies of it we can also construct an infinite family of polytopes in fixed dimension d with increasing number n of facets and of diameter $(1+\epsilon)(n-d)$, for a positive constant ϵ.

The ellipsoid method

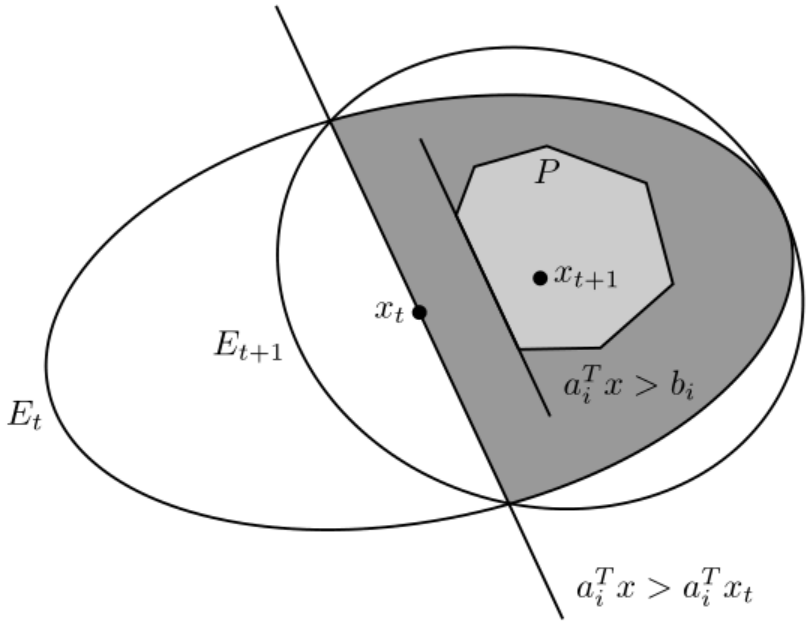


Figure 6: A step of the ellipsoid method where $x_t \notin P$ but $x_{t+1} \in P$. The polytope P and the half-ellipsoid that contains it are shaded.

Theorem 6.2. *Let $E = E(z, D)$ be an ellipsoid in $\mathbb{R}^n$ and $a \in \mathbb{R}^n$ non-zero. Consider the half-space $H = \{x \in \mathbb{R}^n : a^T x \geq a^T z\}$, and let*

$$z' = z + \frac{1}{n+1} \frac{Da}{\sqrt{a^T Da}},$$
$$D' = \frac{n^2}{n^2 - 1} \left(D - \frac{2}{n+1} \frac{Daa^T D}{a^T Da} \right).$$

Then D' is symmetric and positive definite, and therefore $E' = E(z', D')$ is an ellipsoid. Moreover, $E \cap H \subseteq E'$ and $\mathrm{Vol}(E') < e^{-1/(2(n+1))} \mathrm{Vol}(E)$.

Proof sketch of Theorem 6.2. We prove the theorem for $E = \{x \in \mathbb{R}^n : x^T x \leq 1\}$ and $H = \{x \in \mathbb{R}^n : x_1 \geq 0\}$. Since every pair of an ellipsoid and a hyperplane as in the statement of the theorem can be obtained from E and H via some affine transformation, the general case then follows by observing that affine transformations preserve inclusion and, by Lemma 6.3, relative volume of sets.

Let $e_1 = (1, 0, \ldots, 0)^T$. Then,

$$\begin{aligned} E' &= E\left(\frac{e_1}{n+1}, \frac{n^2}{n^2-1}\left(I - \frac{2}{n+1}e_1 e_1^T\right)\right) \\ &= \left\{x \in \mathbb{R}^n : \frac{n^2-1}{n^2}\sum_{i=1}^{n} x_i^2 + \frac{1}{n^2} + \frac{2(n+1)}{n^2}x_1(x_1-1) \leq 1\right\}. \end{aligned}$$

Consider an arbitrary $x \in E \cap H$, and observe that $0 \leq x_1 \leq 1$ and $\sum_{i=1}^{n} x_i^2 \leq 1$. It is easily verified that $x \in E'$ and thus $E \cap H \subseteq E'$.

Now consider the affine transformation $F : \mathbb{R}^n \to \mathbb{R}^n$ given by

$$F(x) = \frac{e_1}{n+1} + \left(\frac{n^2}{n^2-1}\left(I - \frac{2}{n+1}e_1 e_1^T\right)\right)^{\frac{1}{2}} x.$$

It is not hard to show that $E' = F(E)$. Therefore, by Lemma 6.3,

$$\begin{aligned} \frac{\mathrm{Vol}(E')}{\mathrm{Vol}(E)} &= \sqrt{\det\left(\frac{n^2}{n^2-1}\left(I - \frac{2}{n+1}e_1 e_1^T\right)\right)} \\ &= \left(\frac{n^2}{n^2-1}\right)^{\frac{n}{2}}\left(1 - \frac{2}{n+1}\right)^{\frac{1}{2}} = \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{\frac{n-1}{2}} \\ &= \left(1 - \frac{1}{n+1}\right)\left(1 + \frac{1}{n^2-1}\right)^{\frac{n-1}{2}} < e^{-\frac{1}{n+1}}\left(e^{\frac{1}{n^2-1}}\right)^{\frac{n-1}{2}} = e^{-\frac{1}{2(n+1)}}, \end{aligned}$$

where the strict inequality follows by using twice that $1 + a < e^a$ for all $a \neq 0$. □

Ellipsoid method

7.1 Ellipsoid method

Consider a polytope $P = \{x \in \mathbb{R}^n : Ax \geq b\}$, given by a matrix $A \in \mathbb{Z}^{m \times n}$ and a vector $b \in \mathbb{Z}^m$. Assume for now that P is bounded and either empty or full-dimensional. Here, P is called **full-dimensional** if $\mathrm{Vol}(P) > 0$. The ellipsoid method takes the following steps to decide whether P is non-empty:

1. Let U be the largest absolute value among the entries of A and b, and define
   $x_0 = 0, \quad D_0 = n(nU)^{2n} I, \quad E_0 = E(x_0, D_0),$
   $V = (2\sqrt{n})^n (nU)^{n^2}, \quad v = n^{-n}(nU)^{-n^2(n+1)},$
   $t^* = \lceil 2(n+1)\log(V/v) \rceil.$
2. For $t = 0, \ldots, t^*$, do the following:
   1. If $t = t^*$ then stop; P is empty.
   2. If $x_t \in P$ then stop; P is non-empty.
   3. Find a violated constraint, i.e. a row j such that $a_j^T x_t < b_j$.
   4. Let $E_{t+1} = E(x_{t+1}, D_{t+1})$ with

$$x_{t+1} = x_t + \frac{1}{n+1} \frac{D_t a_j}{\sqrt{a_j^T D_t a_j}},$$

$$D_{t+1} = \frac{n^2}{n^2 - 1} \left(D_t - \frac{2}{n+1} \frac{D_t a_j a_j^T D_t}{a_j^T D_t a_j} \right).$$

Karmarkar's algorithm

From Wikipedia, the free encyclopedia

Karmarkar's algorithm is an algorithm introduced by Narendra Karmarkar in 1984 for solving linear programming problems. It was the first reasonably efficient algorithm that solves these problems in polynomial time. The ellipsoid method is also polynomial time but proved to be inefficient in practice.

Where n is the number of variables and L is the number of bits of input to the algorithm, Karmarkar's algorithm requires $O(n^{3.5}L)$ operations on $O(L)$ digit numbers, as compared to $O(n^6L)$ such operations for the ellipsoid algorithm. The runtime of Karmarkar's algorithm is thus

$$O(n^{3.5}L^2 \cdot \log L \cdot \log\log L)$$

using FFT-based multiplication (see Big O notation).

Karmarkar's algorithm falls within the class of interior point methods: the current guess for the solution does not follow the boundary of the feasible set as in the simplex method, but it moves through the interior of the feasible region, improving the approximation of the optimal solution by a definite fraction with every iteration, and converging to an optimal solution with rational data.[1]

Patent controversy - *Can Mathematics be patented?* [edit]

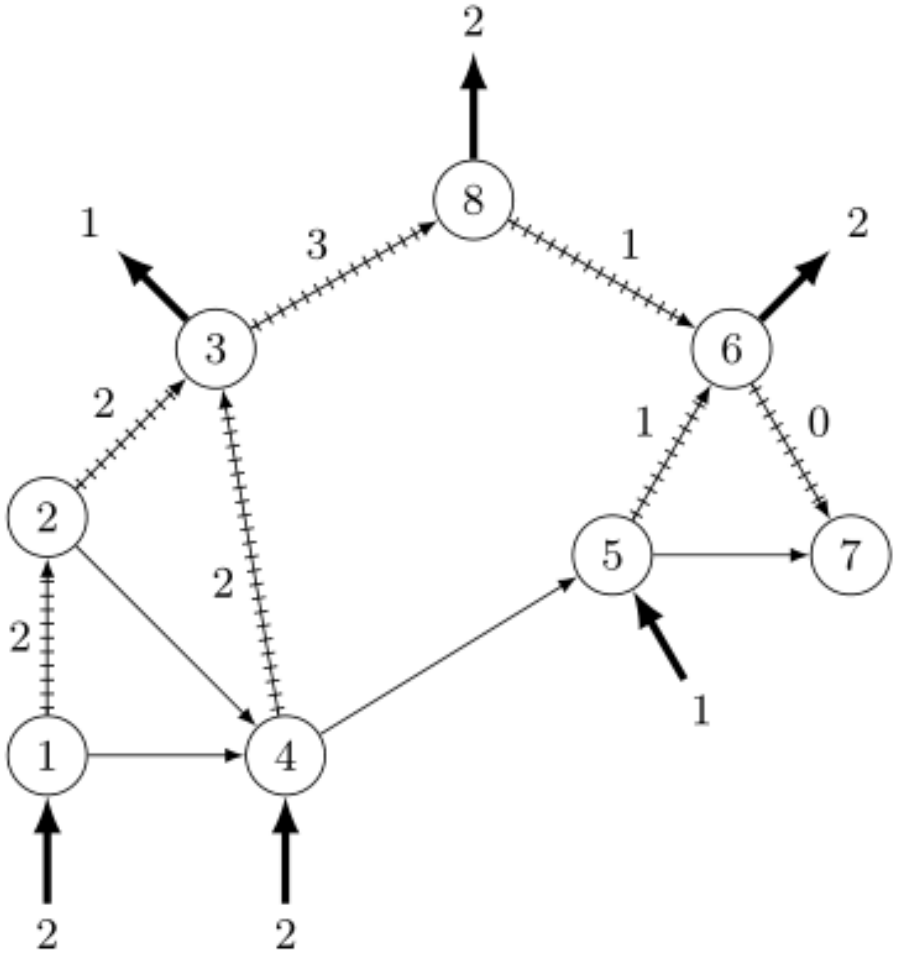
At the time he invented the algorithm, Narendra Karmarkar was employed by AT&T. After applying the algorithm to optimizing AT&T 's telephone network,[14] they realized that his invention could be of practical importance. In April 1985, AT&T promptly applied for a patent on Karmarkar's algorithm and that became more fuel for the ongoing controversy over the issue of software patents.[15] This left many mathematicians uneasy, such as Ronald Rivest (himself one of the holders of the patent on the RSA algorithm), who expressed the opinion that research proceeded on the basis that algorithms should be free. Even before the patent was actually granted, some claimed that there might have been prior art that was applicable.[16]

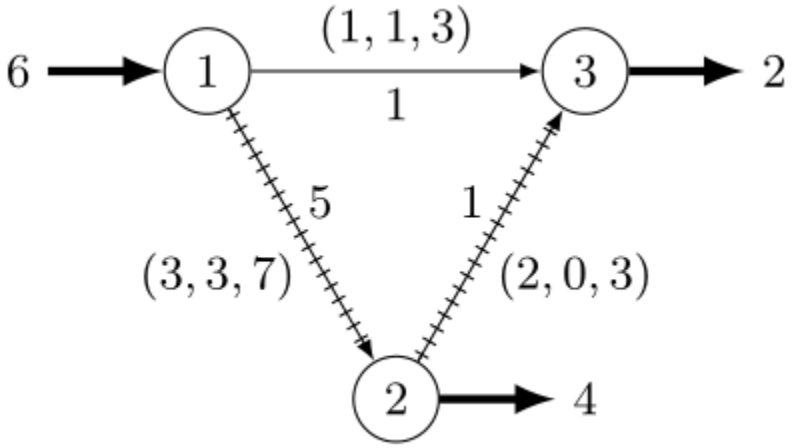
Mathematicians who specialize in numerical analysis such as Philip Gill and others claimed that Karmarkar's algorithm is equivalent to a projected Newton barrier method with a logarithmic barrier function, if the parameters are chosen suitably.[17] However, Gill's argument is flawed, insofar as the method they describe does not even qualify as an "algorithm", since it requires choices of parameters that don't follow from the internal logic of the method, but rely on external guidance, essentially from Karmarkar's algorithm.[18] Furthermore, Karmarkar's contributions are considered far from obvious in light of all prior work, including Fiacco-McCormick, Gill and others cited by Saltzman.[18][19][20] The patent was debated in the U.S. Senate and granted in recognition of the essential originality of Karmarkar's work, as U.S. Patent 4,744,026: "Methods and apparatus for efficient resource allocation" in May 1988. AT&T supplied the KORBX system[21] [22] based on this patent to the Pentagon,[23][24] which enabled them to solve mathematical programming problems which were previously considered unsolvable.

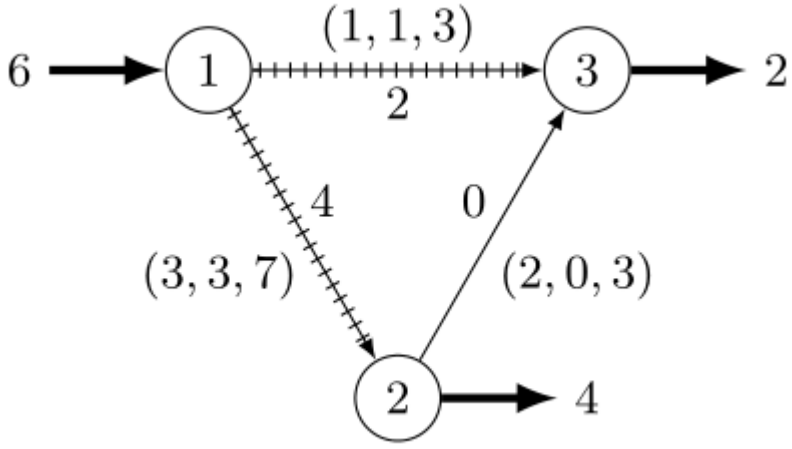
Opponents of software patents have further alleged that the patents ruined the positive interaction cycles that previously characterized the relationship between researchers in linear programming and industry, and specifically it isolated Karmarkar himself from the network of mathematical researchers in his field. [25]

The patent itself expired in April 2006, and the algorithm is presently in the public domain.

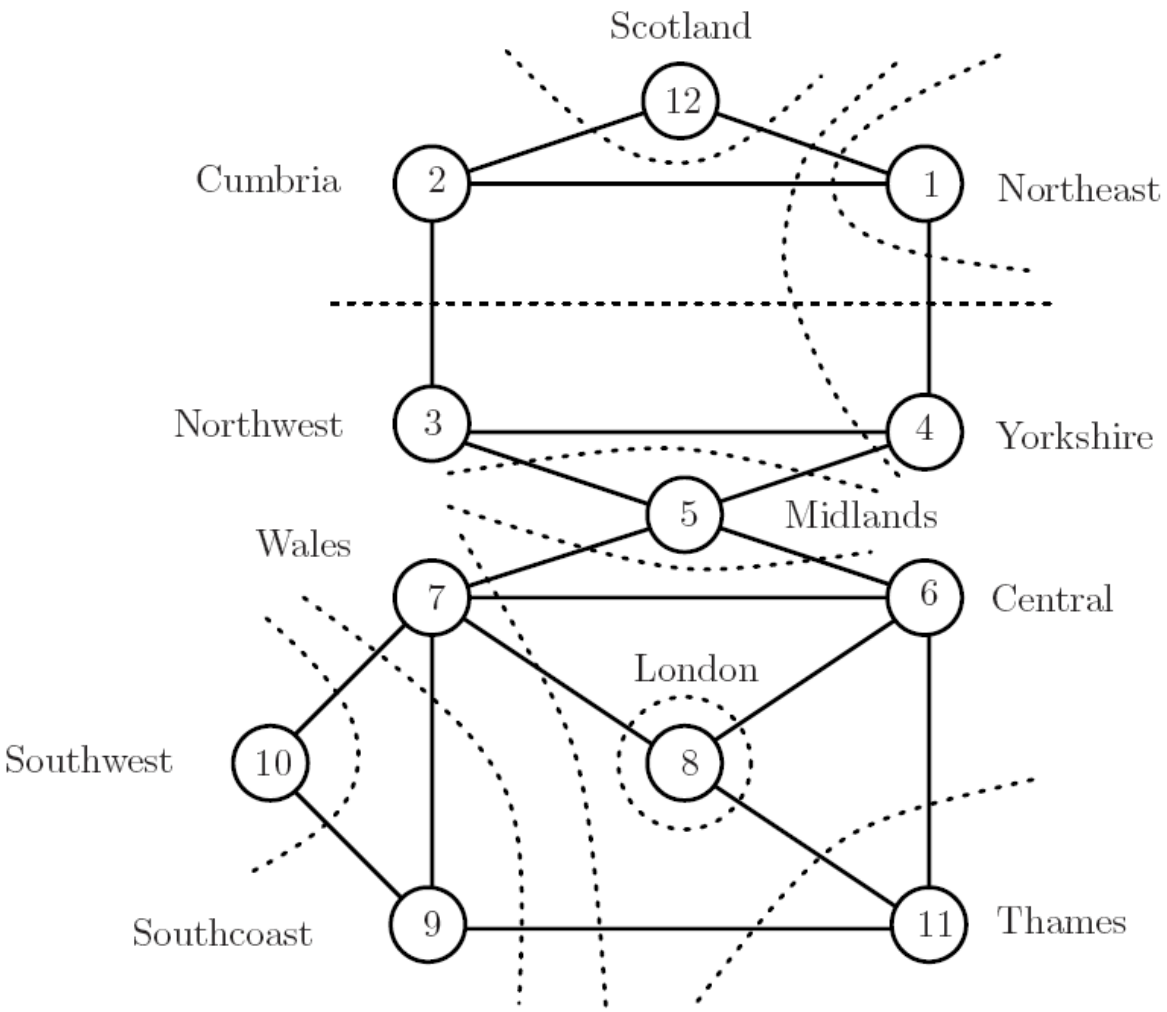
A basic feasible flow is a spanning tree




6
1
(1, 1, 3)
1
3
2
5
(3, 3, 7)
1
(2, 0, 3)
2
4


6
1
(1, 1, 3)
2
3
2
4
(3, 3, 7)
0
(2, 0, 3)
2
4

Network simplex algorithm



The demand for electricity at node i is d_i. Node i has k_i generators, that can generate electricity at costs of $a_{i1}, \ldots, a_{ik_i}$, up to amounts $b_{i1}, \ldots, b_{ik_i}$. There are $n = 12$ nodes and 351 generators in all. The capacity for transmission from node i to j is c_{ij} $(= c_{ji})$.

Let x_{ij} = amount of electricity carried $i \to j$ and let y_{ij} = amount of electricity generated by generator j at node i. The LP is

$$\text{minimize} \sum_{ij} a_{ij} y_{ij}$$

$$\text{subject to} \sum_j y_{ij} - \sum_j x_{ij} + \sum_j x_{ji} = d_i, \quad i = 1, \ldots, 12,$$

$$0 \le x_{ij} \le c_{ij}, \ 0 \le y_{ij} \le b_{ij}.$$

In addition, there are constraints on the maximum amount of power that may be shipped across the cuts shown by the dotted lines in the diagram.