§5 Resource allocation

Note Title 14/10/2011

The "job" of a distributed network is to decide who gets to use how much capacity. It may be by end-system adaptation (TCP), or by explicit admission control (telephone system tells you "no circuits available"), or by implicit admission control (your http request gets dropped because the server is overloaded).

It's worth thinking about resource allocation as a general principle, because

- you'll have to address it sooner or later
  e.g. you've built an amazing new public transport infrastructure so big that there's is no congestion at all. But.... three years later is overcrowded, and you wish you had put in place mechanisms to ensure fair access

- it's hard to get right, except in simple cases
  A single congested link? Share it evenly between all flows.
  A network, where different users use different routes, and have different hop counts, and some flows are corporate VPNs (many people inside one flow) and other flows are BitTorrent (many flows for one person). Tricky!

- There is a systematic way to formulate & solve the problem
  1. identify the feasible region
  2. choose a weighted α-fair operating point
  3. crank out a distributed algorithm for reaching that point.

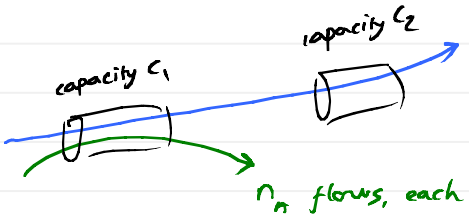
  And it's the only general solution we know.

- Once you understand the systematic principle, you can apply it "horizontally" across many systems.
  - which bits of TCP are generic to any resource allocation problem?
  - if I apply TCP-style allocation to admission control for a distributed hash table, what do I get?
  - which bits of TCP are systems tweaks for making the general principle actually work in practice?

§ 5.1 The feasible region

An allocation vector $(y_1, \cdots, y_n)$ specifies the flow rates that each user gets. The feasible region is the set of allocation vectors that can be supported, given the available capacity in a system.

EXAMPLE 1. TCP.



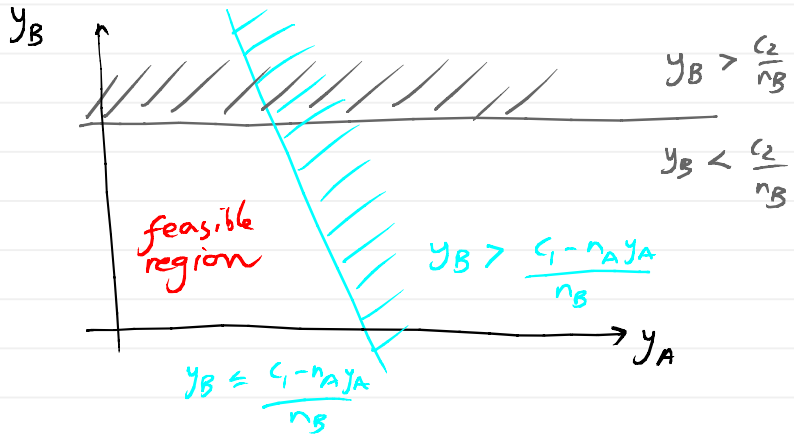
Goodput = amount of traffic that actually gets through

Offered load = amount of traffic sent

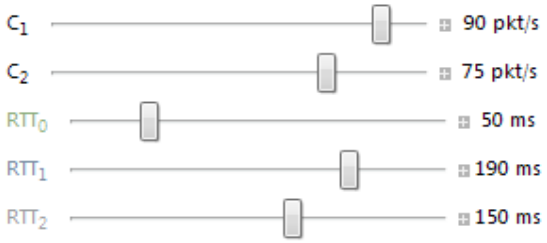
An allocation vector (y_A, y_B) is feasible if neither link is overloaded, ie

$$n_A y_A + n_B y_B \leq C_1$$
and $$n_B y_B \leq C_2$$

The feasible region is $\{ (y_A, y_B) : n_A y_A + n_B y_B \leq C_1 \text{ and } n_B y_B \leq C_2 \}$

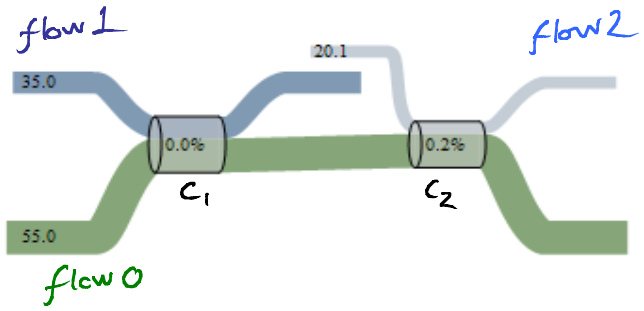


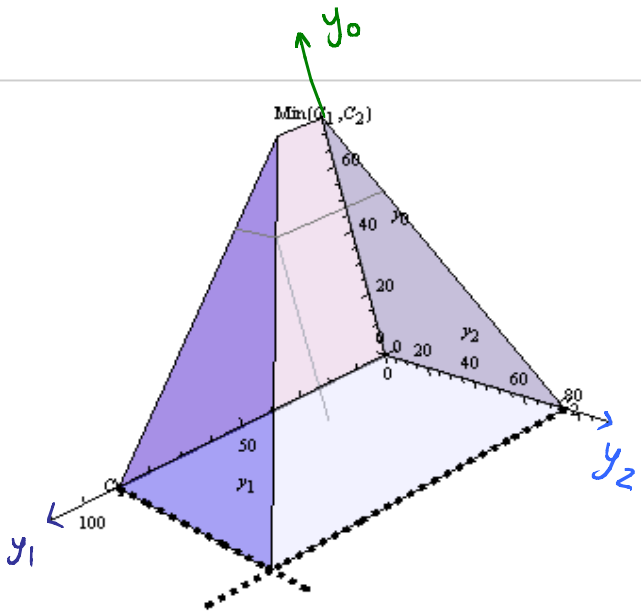
EXAMPLE 1b. TCP.



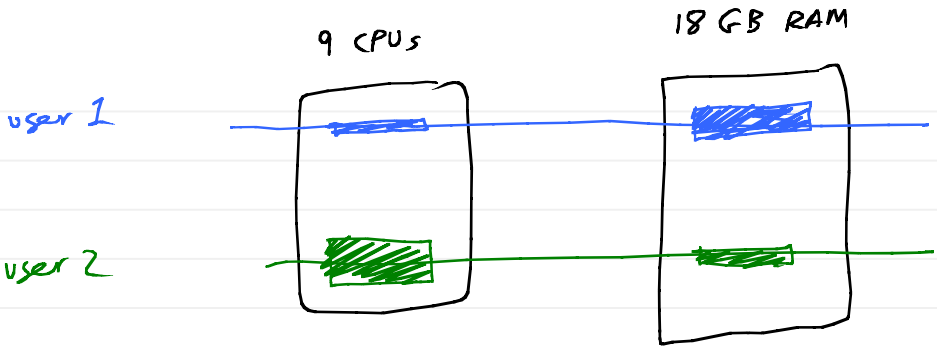
Feasible region:

2 links, 3 flows:





EXAMPLE 2. Data Centre.



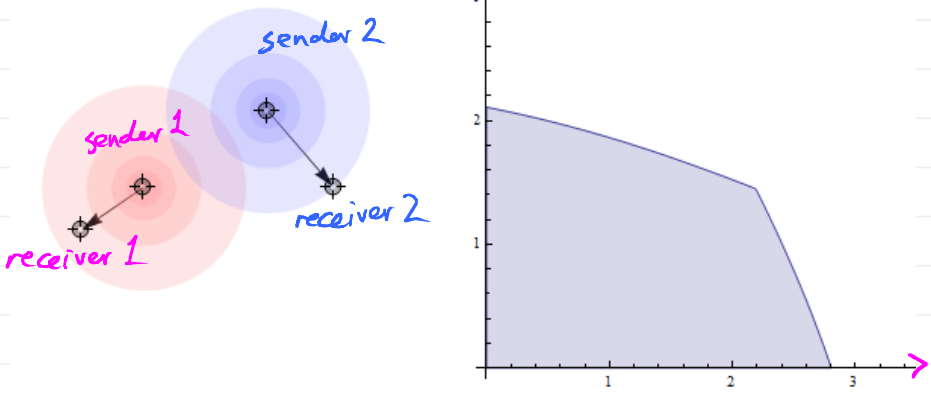
one job requires 1 CPU and 4GB RAM to complete in 1 min

one job requires 3 CPU and 1GB RAM to complete in 1 min.

Let y_1 = no. of "instances" assigned to user 1 = # jobs completed/min
y_2 = ——— 2 ———.

The feasible region is $\{(y_1, y_2): \quad y_1 + 3y_2 \le 9 \quad \text{(CPU constraint)}, \quad 4y_1 + y_2 \le 18 \quad \text{(RAM constraint)}\}$.

EXAMPLE 3. Wireless capacity.



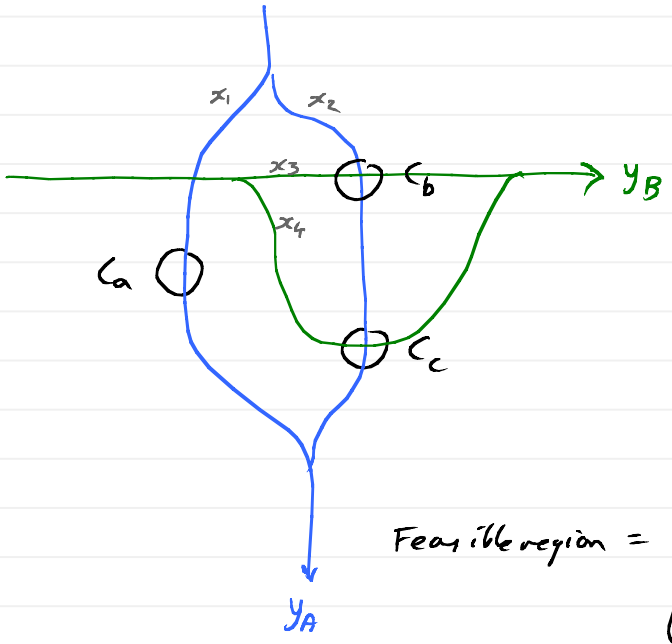
Suppose there are two transmitters and two receivers, in given positions, with given levels of background noise, and given max. transmit power.
Let user 1 choose to transmit at a fraction x_1 of max power, user 2 ——— x_2 ———.

Information theory tells us:
user 1 gets throughput $B \log_2\left(1 + \frac{x_1 h_{1,1} P_1}{N + x_2 h_{2,1} P_2}\right)$, user 2 gets $B \log_2\left(1 + \frac{x_2 h_{2,2} P_2}{N + x_1 h_{1,2} P_1}\right)$.

So, feasible region = $\{(y_1, y_2):$ there exist x_1, x_2 such that $y_1 \le B \log_2(\cdots), \quad y_2 \le B \log_2(\cdots)\}$.

Cleverer decoding procedures (successive interference cancellation) correspond to different formulae for y_1 & y_2.

EXAMPLE 4. Multipath TCP.

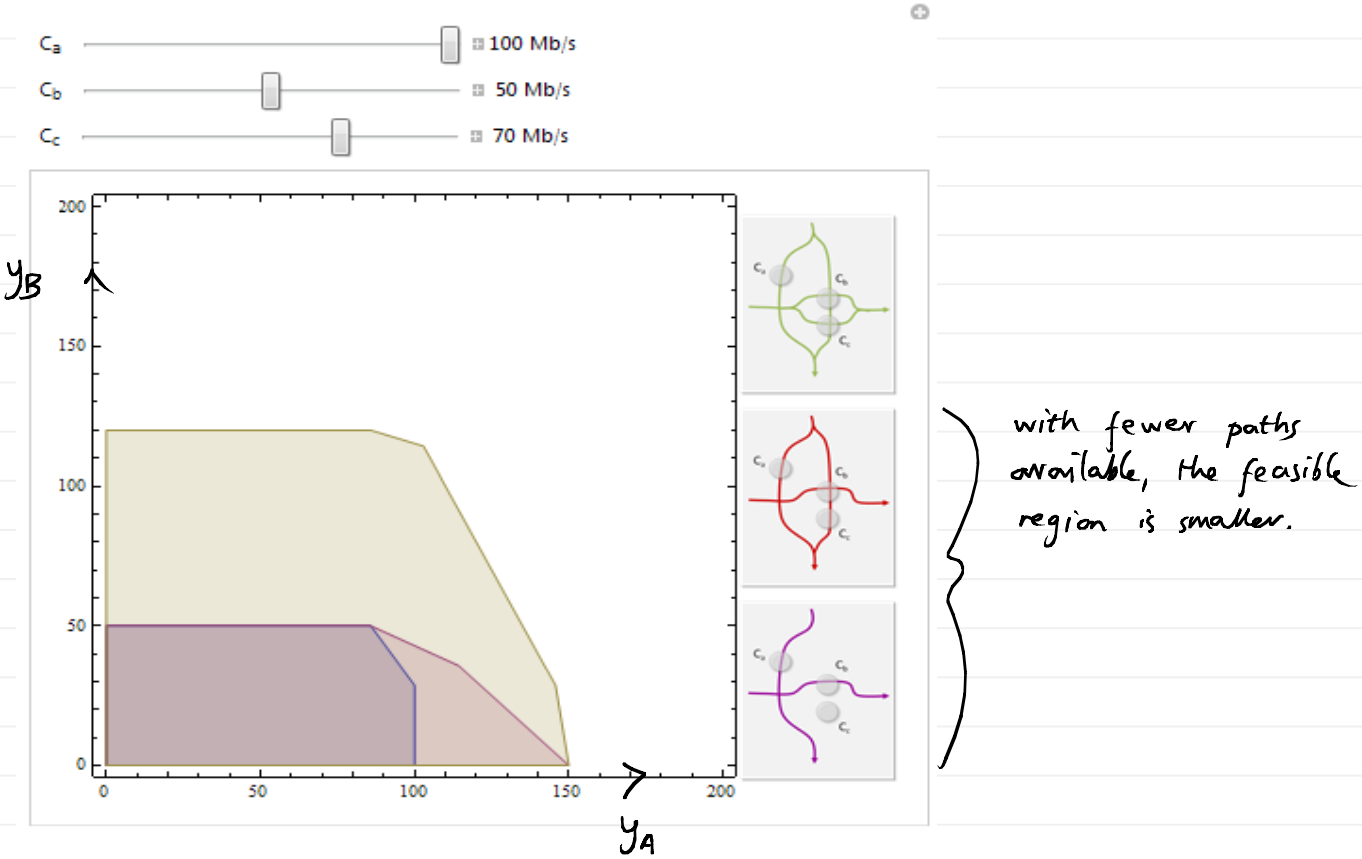


Suppose there are two flows:
- flow A has goodput x_1 on left-hand path and x_2 on right-hand path
- flow B has goodput x_3 on top path and x_4 on bottom path.

This is feasible if $x_1 \le C_a$
$x_2 + x_3 \le C_b$
$x_4 + x_2 \le C_c$.

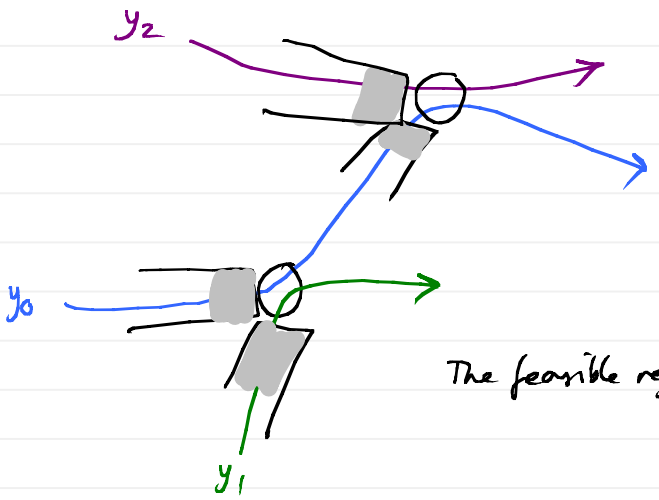
Feasible region = $\{(y_A, y_B)$: there exist x_1, x_2, x_3, x_4 such that
$y_A = x_1 + x_2$, $y_B = x_3 + x_4$,
$x_1 \le C_a$, $x_2 + x_3 \le C_b$, $x_4 + x_2 \le C_c\}$

This turns out to be $\{(y_A, y_B)$: $y_A \le C_a + \min(C_b, C_c)$
$y_B \le C_b + C_c$
$2y_A + y_B \le 2C_a + C_b + C_c\}$



with fewer paths available, the feasible region is smaller.

EXAMPLE 5. Admission control for a distributed algorithm



Two servers,
three request types.

Each server can handle C jobs/sec.

What arrival rates can be supported?

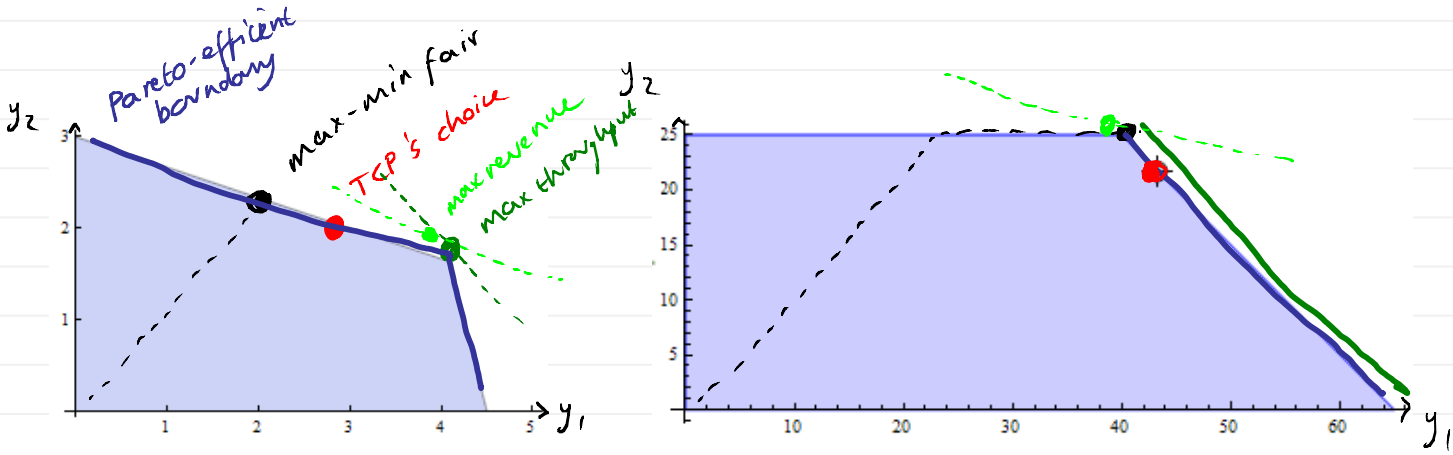
The feasible region is $F = \left\{ (y_0, y_1, y_2) : \begin{array}{r} y_0 + y_1 \le C \\ y_0 + y_2 \le C \end{array} \right\}$.

If requests arrive at rates $(\lambda_0, \lambda_1, \lambda_2) \in F$, then all requests can be served.

If requests arrive at rates $(\lambda_0, \lambda_1, \lambda_2) \notin F$, then some requests must be dropped, and the actual served job rates must be some $(y_0, y_1, y_2) \in F$.
The drop rates are $\left(\frac{\lambda_0 - y_0}{\lambda_0}, \frac{\lambda_1 - y_1}{\lambda_1}, \frac{\lambda_2 - y_2}{\lambda_2} \right)$.

§ 5.2 Choosing the operating point

The system has to operate at some point in the feasible region. Which?



There are arguments to be made in favour of all of these.

It turns out that every one of these goals is equivalent to solving for a suitable choice of weights $w_s > 0$ and $\alpha \in [0, \infty)$, the optimization problem

$$\text{maximize} \quad \sum_s w_s \frac{y_s^{1-\alpha}}{1-\alpha} \quad \text{over all feasible allocation vectors } y$$

* If $\alpha = 1$, use $\sum_s w_s \log(y_s)$ instead.

called the weighted α-fair utility function.

e.g. $\text{maximize} \quad \sum_s \frac{2}{RTT_s^2} \frac{y_s^{-2}}{-1}$ over all feasible y is what TCP does

$\lim_{\alpha \to \infty} \max. \sum_s \frac{y_s^{1-\alpha}}{1-\alpha}$ over all feasible y is max-min fair.

$\text{maximize} \quad \sum_s y_s$ over all feasible y is maximum throughput

$\text{maximize} \quad \sum_s r_s y_s$ over all feasible y is maximum revenue (where y_s pays at rate r_s)

To understand how it is that TCP solves the constrained optimization, we first need to learn how to solve constrained optimization problems...

THOUGHTS ABOUT TCP AS AN OPTIMIZATION-SOLVER

Jacobson had no intention to solve this optimization problem —— but nonetheless it is there, hidden in his code. I call this TELEOLOGY. which means "study of the intention or ultimate purpose of actions".

- It's generally hard to find distributed algorithms to solve complicated optimization problems. But here, we have a worldwide optimization problem, which is successfully distributed over every single computer connected to the Internet.

- This optimization problem can be shown to have a unique solution. Therefore TCP has only one fixed point, so it's not bistable like Dynamic Alt. Routing. Often, if a distributed system solves a well-behaved optimization problem, then the distributed system is likely to behave well. Also, whereas the iterative fixed point method and the drift model method may sometimes run into problems (ie the values keep jumping around when you try to solve them), there are robust methods for solving optimization problems of this sort.

The optimization problem is called by economists a "social welfare optimization".

maximize $\sum_s U_s(y_s)$ over all feasible y

$U_s(y_s)$: Utility or "happiness" of individual s with his/her allocation y_s

Jeremy Bentham (1748–1832)
Radical political theorist, secularist, founder of UCL, and father of the political/social theory of utilitarianism:

"The good is whatever brings the greatest happiness to the greatest number of people."

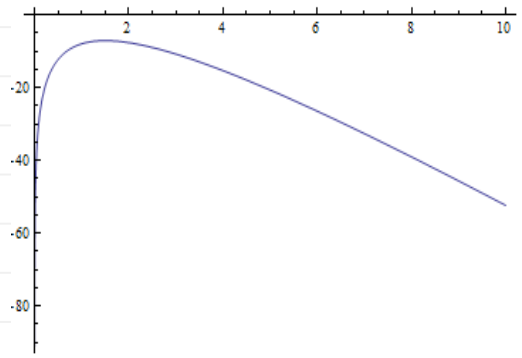
In effect, The TCP algorithm has chosen utility function

$$U_s(y_s) = \frac{-2}{y_s RTT_s^2}.$$

Is this a good measure of user happiness?
If we preferred a different measure, could we redesign TCP to achieve it?

§ 5.3 Constrained multivariate optimization

HOW TO SOLVE A 1D OPTIMIZATION



Given a function like

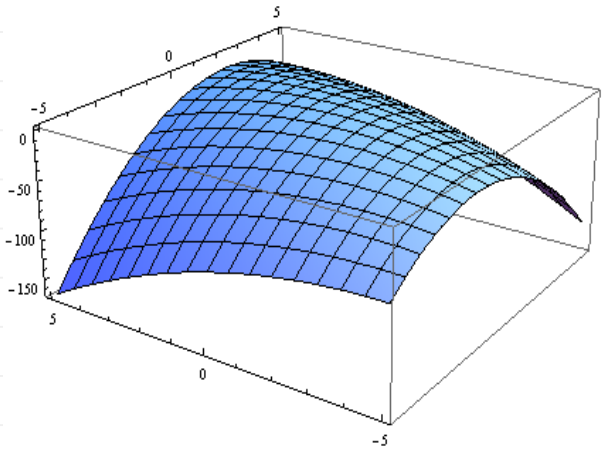
$$f(x) = 12 \log x - 10x$$

you maximize it by solving $\frac{df}{dx} = 0$:

$$\frac{12}{x} - 10 = 0 \Rightarrow x = \frac{12}{10}.$$

(and check that this is indeed a local max, and that there are no bigger local maxes.)

HOW TO SOLVE A 2D OPTIMIZATION

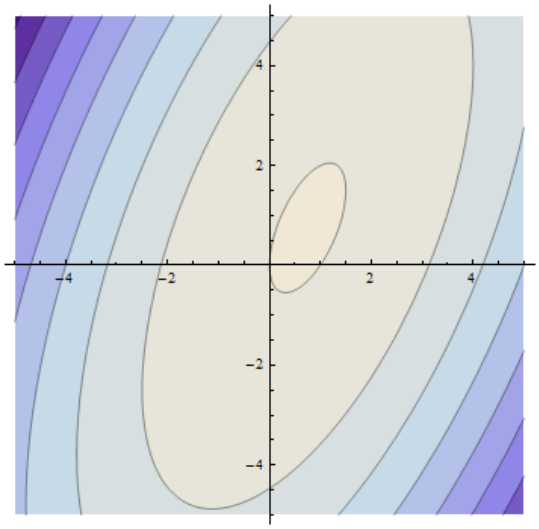


Given a function like

$$f(x,y) = 2xy + 3x - 3x^2 - y^2$$

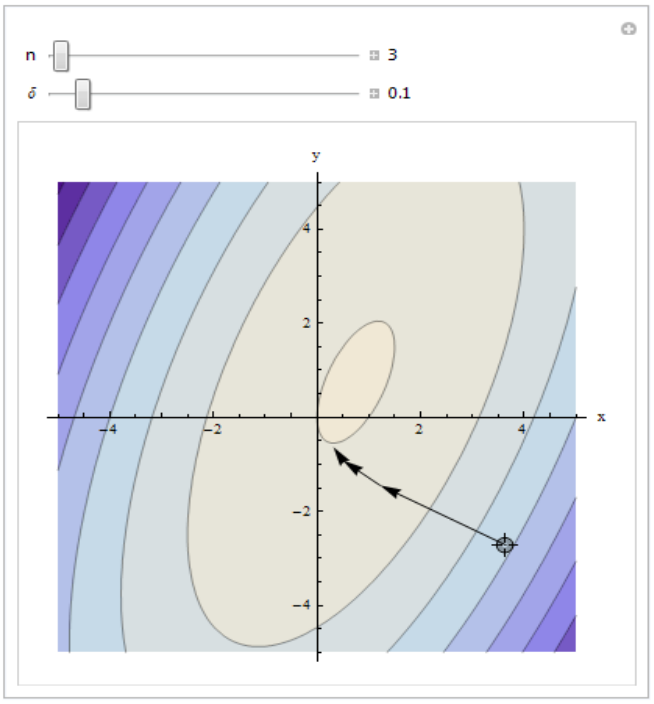
you maximize it by solving $\frac{\partial f}{\partial x} = 0$ and $\frac{\partial f}{\partial y} = 0$:

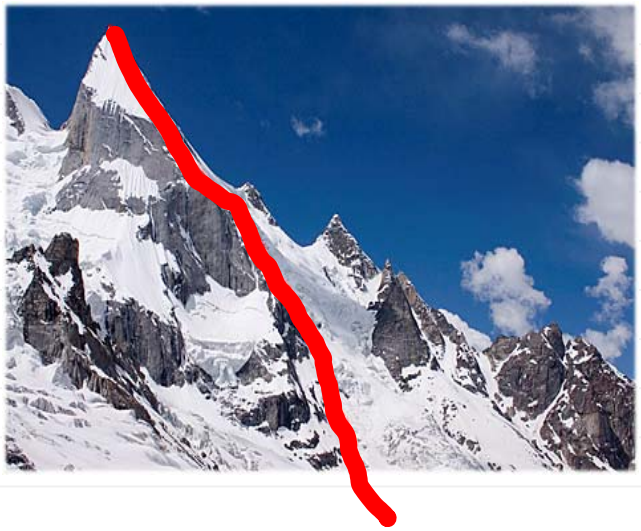
$$\left.\begin{aligned} \frac{\partial f}{\partial x} &= 2y + 3 - 6x = 0 \\ \frac{\partial f}{\partial y} &= 2x - 2y = 0 \end{aligned}\right\} \Rightarrow x = y = \frac{3}{4}.$$



Sometimes, a contour plot shows us more clearly the shape of the function.

HOW TO SOLVE A 2D OPTIMIZATION IF YOU'RE A COMPUTER / HIKER





Take step after step in the direction of steepest ascent / gradient.
In order not to overshoot, take smaller steps as you reach the summit — you can achieve this by letting step size be proportional to gradient, if the function is smooth at the summit

e.g. $f(x,y) = 2xy + 3x - 3x^2 - y^2 \Rightarrow \frac{\partial f}{\partial x} = 2y+3-6x, \quad \frac{\partial f}{\partial y} = 2x-2y.$

The steepest ascent rule says: update your guess (x,y) by

$$(x,y) \leftarrow (x,y) + \delta\,(2y+3-6x,\ 2x-2y)$$

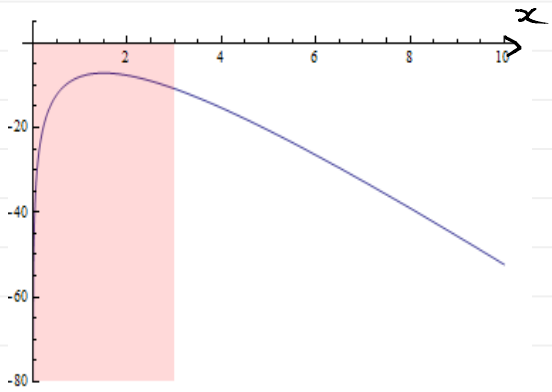
You have to pick δ appropriately, to avoid overshoot (δ too large) and creeping (δ too small).

This is a drift model. We could write it as

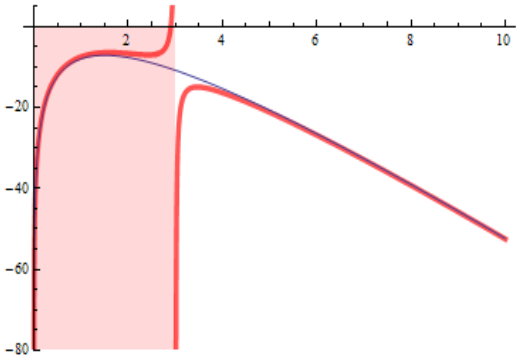
$$\text{drift in } x = \frac{dx}{dt} = 2y+3-6x$$

$$\text{drift in } y = \frac{dy}{dt} = 2x-2y.$$

HOW TO SOLVE A CONSTRAINED 1D OPTIMIZATION



e.g. maximize $f(x) = 12\log(x) - 10x$
such that $x \geq 3$



A simple trick is to "relax" the constraint by turning it into a penalty,
e.g.

$$\text{maximize } f(x) = 12\log(x) - 10x - \frac{\varepsilon}{x-3}$$

if x is just a bit larger than 3, this penalty becomes very large -ve.

To max. this, solve $\frac{df}{dx} = \frac{12}{x} - 10 + \frac{\varepsilon}{(x-3)^2} = 0$

At $\varepsilon = .1$: opt at $x = 1.20,\ 2.87,\ 3.13$
$\varepsilon = .01$, opt at $x = 1.20,\ 2.96,\ 3.04$
$\varepsilon = .001$, opt at $x = 1.20,\ 2.99,\ 3.01$.

We'd pick ε small-ish, and we'd select the solution that lies in the range we want ($x \geq 3$).

HOW TO SOLVE A CONSTRAINED 2D OPTIMIZATION BY COMPUTER

e.g. Maximize $f(x,y) = 2xy + 3x - 3x^2 - y^2$

such that $x+y \le 1$ and $y \le 0.6$.

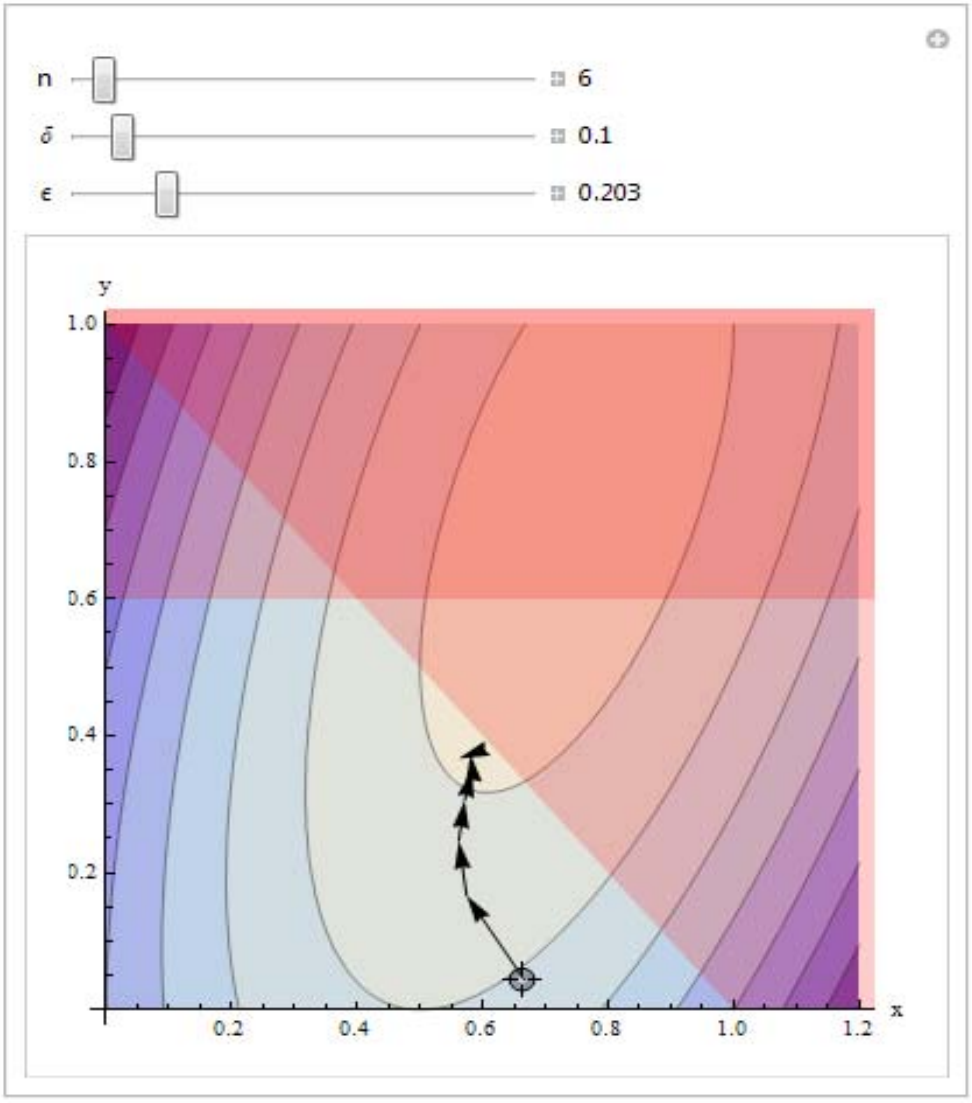
We first relax the constraints by inventing penalty functions,

e.g. constraint $y \le 0.6$: penalty function $\frac{\varepsilon}{0.6-y}$

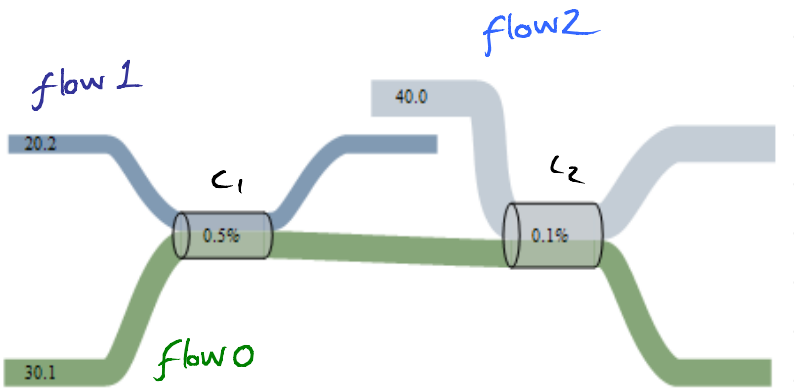
constraint $x+y \le 1$: penalty function $\frac{\varepsilon}{1-(x+y)}$

Then we solve the relaxed problem

maximize $2xy + 3x - 3x^2 - y^2 - \frac{\varepsilon}{0.6-y} - \frac{\varepsilon}{1-(x+y)}$



§ 5.4 Solving the resource allocation problem



Example: maximize $\frac{-2}{RTT_0^2 y_0} - \frac{2}{RTT_1^2 y_1} - \frac{2}{RTT_2^2 y_2}$

such that $y_0 + y_1 \le C_1$ and $y_0 + y_2 \le C_2$.

To solve this by computer, we first invent penalty functions, say $L_1(y_0+y_1)$ and $L_2(y_0+y_2)$. Then pick y_0, y_1, y_2 to maximize

$$f(y_0, y_1, y_2) = -\frac{2}{RTT_0^2 y_0} - \frac{2}{RTT_1^2 y_1} - \frac{2}{RTT_2^2 y_2} - L_1(y_0+y_1) - L_2(y_0+y_2).$$

We solve it by drifting in the direction of steepest ascent. The gradient is

$$\frac{\partial f}{\partial y_0} = \frac{2}{RTT_0^2 y_0^2} - L_1'(y_0+y_1) - L_2'(y_0+y_2)$$

$$\frac{\partial f}{\partial y_1} = \frac{2}{RTT_1^2 y_1^2} - L_1'(y_0+y_1) \quad \leftarrow \text{call this } p_1$$

$$\frac{\partial f}{\partial y_2} = \frac{2}{RTT_2^2 y_2^2} - L_2'(y_0+y_2). \quad \leftarrow \text{call this } p_2$$

To solve it by computer, we should solve the drift model

$$\frac{dy_0}{dt} = \frac{2}{RTT_0^2 y_0^2} - p_1 - p_2, \quad \frac{dy_1}{dt} = \frac{2}{RTT_1^2 y_1^2} - p_1, \quad \frac{dy_2}{dt} = \frac{2}{RTT_2^2 y_2^2} - p_2.$$

i.e. pick a step size δ and repeatedly do

$y_0 \leftarrow y_0 + \delta \times$ drift in y_0, $y_1 \leftarrow y_1 + \delta \times$ drift in y_1, $y_2 \leftarrow y_2 + \delta \times$ drift in y_2.

Let's think of this in terms of window sizes $w_0 = y_0 RTT_0$, etc.:

$$\frac{dw_0}{dt} = RTT_0 \frac{dy_0}{dt} = RTT_0 \left(\frac{2}{RTT_0^2 y_0^2} - p_1 - p_2 \right) = RTT_0 \left(\frac{2}{w_0^2} - p_1 - p_2 \right) \quad \text{etc.}$$

Clearly the fixed point of this 3-dimensional drift model is not affected if we scale the drifts, even if we scale them separately:

$$\frac{dw_0}{dt} = \gamma_0 RTT_0 \left(\frac{2}{w_0^2} - p_1 - p_2 \right), \quad \frac{dw_1}{dt} = \gamma_1 RTT_1 \left(\frac{2}{w_1^2} - p_1 \right), \quad \frac{dw_2}{dt} = \gamma_2 RTT_2 \left(\frac{2}{w_2^2} - p_2 \right)$$

I shall pick $\gamma_0 = \frac{w_0^2}{2 RTT_0^2}$, etc., giving

$$\frac{dw_0}{dt} = \frac{1}{RTT_0} - (p_1 + p_2) \frac{w_0^2}{2 RTT_0}, \quad \frac{dw_1}{dt} = \frac{1}{RTT_1} - \frac{p_1 w_1^2}{2 RTT_1}, \quad \frac{dw_2}{dt} = \frac{1}{RTT_2} - \frac{p_2 w_2^2}{2 RTT}.$$

These are exactly the drift equations for TCP!

In other words, TCP <u>is</u> the steepest-ascent numerical method for solving the α-fair resource allocation problem!

What are p_1 and p_2?
From the perspective of the optimization problem, what I need is some arbitrary penalty function $L_1(y_0+y_1)$ which grows as y_0+y_1 approaches C_1, and I defined $p_1 = L_1'(y_0+y_1)$. So, all I need is $p_1 \approx 0$ if $y_0+y_1 \ll C_1$ and p_1 large if $y_0+y_1 \approx C_1$. The drop probability at link 1 is exactly such a function. The exact drop probability function (depending on e.g. buffer size) will determine the exact penalty function. These sorts of optimization problems tend to be easier to compute if the penalty function is reasonably "gentle".

NOTES.

TCP works by increasing its window when it gets an ACK, decreasing its window when it detects a drop. The increase/decrease rules lead us to the drift model. But we can also go the other way round: we could let the increase be some arbitrary function $i(w)$, and let the decrease be some arbitrary function $d(w)$, and ask: what functions $i(w)$ and $d(w)$ yield a desired drift equation?

What's the purpose of that funny rescaling with γ? It's a kludge, to get this optimization approach to match up with what TCP actually does. It doesn't change the fixed point (i.e. the equilibrium throughput for each flow), it only changes the speed of getting there. In fact, what TCP actually does is BAD: its drift is too small for some w, too large for other w.

We (at UCL) invented multipath congestion control for TCP by simply applying this optimization procedure to the multipath feasible region — and then working hard to get all the kludge fixes right. We wouldn't have known where to start without this.