Air courier

The air branch of a shipping company uses a fleet of Boeing 777s and 747s cargo to serve the EMEA demands. A 777 can carry 653 m^3 in volume and 103 tonnes (t) in weight. A 747 can carry 854.5 m^3 and 134.2 t. Each freighter is dedicated to a single segment (origin to destination airport and back once a day: both flights happen within the same 24 hours). The demand matrix is extremely fine-grained, and consists of all order IDs (packages) for the week, with origin and destination airports, weight and volume. The networks consists of airports, linked by the segments that are actually flown. The per-mile cost of flying is a linearly increasing function of the loaded weight (the two functions are different for 777 and 747). Flights can leave empty (in which case the company subcontracts the flight); company policy states that, if loaded, the loaded volume has to fill at least half the capacity. Model the corresponding variant of the multicommodity flow problem.

Air courier: .mod file

```
## airports
set Airports;
param Dist{Airports, Airports} >= 0, default Uniform(100,2000);
## aircrafts
set AircraftTypes;
# max volume per aircraft type
param AV{AircraftTypes} >= 0;
# max weight per aircraft type
param AW{AircraftTypes} >= 0;
# cost per mile per aircraft type
param ACpM{AircraftTypes} >= 0;
# number of flights on the time horizon
param DaysMax := 7;
set Days := 1..DaysMax;
```

Air courier: .mod file

```
## flight segment network
set Segments within {Airports, Airports};
param Aircraft{Segments} symbolic;
param VolumeCap{(u,v) in Segments} >= 0, default DaysMax*AV[Aircraft[u,v]];
param WeightCap{(u,v) in Segments} >= 0, default DaysMax*AW[Aircraft[u,v]];
param ArcCost{(u,v) in Segments} default ACpM[Aircraft[u,v]]*Dist[u,v];

## fine-grained demand
param Dmax;
set Demand;
param Volume{Demand} >= 0;
param Weight{Demand} >= 0;
param Orig{Demand} symbolic;
param Dest{Demand} symbolic;

## aggregated demand
set D within {Airports,Airports};
param dV{D} >= 0, default 0;
param dW{D} >= 0, default 0;
```

Air courier: .mod file

```
## decision variables

# volume (unsplittable) flow
var V{Segments,D} binary;
# weight (unsplittable) flow
var W{Segments,D} binary;
# whether a flight leaves empty
var E{Segments,Days} binary;

## objective function
minimize cost:
  sum{(h,k) in D, (u,v) in Segments} ArcCost[u,v]*W[u,v,h,k];
```

Air courier: .mod file

```
## constraints
# volume multiflow
subject to volumeFlow{(h,k) in D, v in Airports}:
  sum{w in Airports: (v,w) in Segments} V[v,w,h,k] -
  sum{u in Airports: (u,v) in Segments} V[u,v,h,k] =
    if (v == h) then 1 else if (v == k) then -1 else 0;
subject to volumeCapacity{(u,v) in Segments}:
  sum{(h,k) in D} dV[h,k]*V[u,v,h,k] <= VolumeCap[u,v];

# weight multiflow
subject to weightFlow{(h,k) in D, v in Airports}:
  sum{w in Airports: (v,w) in Segments} W[v,w,h,k] -
  sum{u in Airports: (u,v) in Segments} W[u,v,h,k] =
    if (v == h) then 1 else if (v == k) then -1 else 0;
subject to weightCapacity{(u,v) in Segments}:
  sum{(h,k) in D} dW[h,k]*W[u,v,h,k] <= WeightCap[u,v];
```

Air courier: .mod file

```
# consistency: can't spread the aggregated flow!
subject to consistent{(h,k) in D, (u,v) in Segments}:
  V[u,v,h,k] = W[u,v,h,k];

# company policy on non-empty flights (at least half volume)
subject to companypolicy1{(u,v) in Segments}:
  sum{(h,k) in D} dV[h,k]*V[u,v,h,k] >=
    (0.5*VolumeCap[u,v]/DaysMax)*sum{t in Days} E[u,v,t];
subject to companypolicy2{(u,v) in Segments}:
  sum{(h,k) in D} dV[h,k]*V[u,v,h,k] <=
    (VolumeCap[u,v]/DaysMax)*sum{t in Days} E[u,v,t];
```

Air courier: .run file

```
param eps := 1e-6;
model air_courier.mod;
data airports.dat;
data aircrafts.dat;
data segments.dat;
data demands.dat;

# aggregate the fine-grained demand
let D := {};
param orig symbolic;
param dest symbolic;
for {d in Demand} {
  let orig := Orig[d];
  let dest := Dest[d];
  let D := D union {(orig,dest)};
  let dV[orig,dest] := dV[orig,dest] + Volume[d];
  let dW[orig,dest] := dW[orig,dest] + Weight[d];
}

option solver cplex;
solve;
```

Air courier: .run file

```
param curra symbolic;
param nexta symbolic;
param nnext integer, default 0;
if solve_result == "infeasible" then {
  printf "instance is infeasible\n";
} else {
  for {(h,k) in D} {
    printf "demand [%s,%s]: %s", h,k, h;
    let curra := h;
    repeat while(curra != k) {
      let nnext :=
card({v in Airports: (curra,v) in Segments and abs(V[curra,v,h,k]-1)<eps});
      if (nnext != 1) then {
        printf "ERROR: %d next vtx after %d (check absmipgap)\n",curra,nnext;
        break;
      }
      for {v in Airports:(curra,v) in Segments and abs(V[curra,v,h,k]-1)<eps}{
        let nexta := v;
      }
      printf " -(%d)-> %s", sum{t in Days} E[curra,nexta,t], nexta;
      let curra := nexta;
    }
    printf "\n";
  }
}
```

Subsection 2

Sparsity and ℓ_1 minimization

Coding problem 1

- **Need to send sparse vector $y \in \mathbb{R}^n$ with $n \gg 1$**

1. **Sample full rank $k \times n$ matrix A with $k \ll n$**

   preliminary: both parties know A

2. **Encode $b = Ay \in \mathbb{R}^k$**
3. **Send b**

- **Decode by finding sparsest x s.t. $Ax = b$**

Coding problem 2

- Need to send a sequence $w \in \mathbb{R}^k$
- Encoding $n \times k$ matrix Q, with $n \gg k$, send $z = Qw \in \mathbb{R}^n$

  preliminary: both parties know Q
- (Low) prob. e of error: $e\,n$ comp. of z sent wrong

  they can be totally off
- *Receive (wrong) vector $\bar{z} = z + x$ where x is sparse*
- Can we recover z?
  - Choose $k \times n$ matrix A s.t. $AQ = 0$
  - Let $b = A\bar{z} = A(z + x) = A(Qw + x) = AQw + Ax = Ax$
  - *Suppose we can find sparsest x' s.t. $Ax' = b$*
  - $\Rightarrow$ we can recover $z' = \bar{z} - x'$
- Recover $w' = (Q^\top Q)^{-1} Q^\top z'$

  Likelihood of getting small $\|w - w'\|$?

Sparsest solution of a linear system

- **Problem** $\min\{\|x\|_0 \mid Ax = b\}$ **is NP-hard**

  Reduction from Exact Cover by 3-Sets [Garey&Johnson 1979, A6[MP5]]

- **Relax to** $\min\{\|x\|_1 \mid Ax = b\}$
- **Reformulate to LP:**

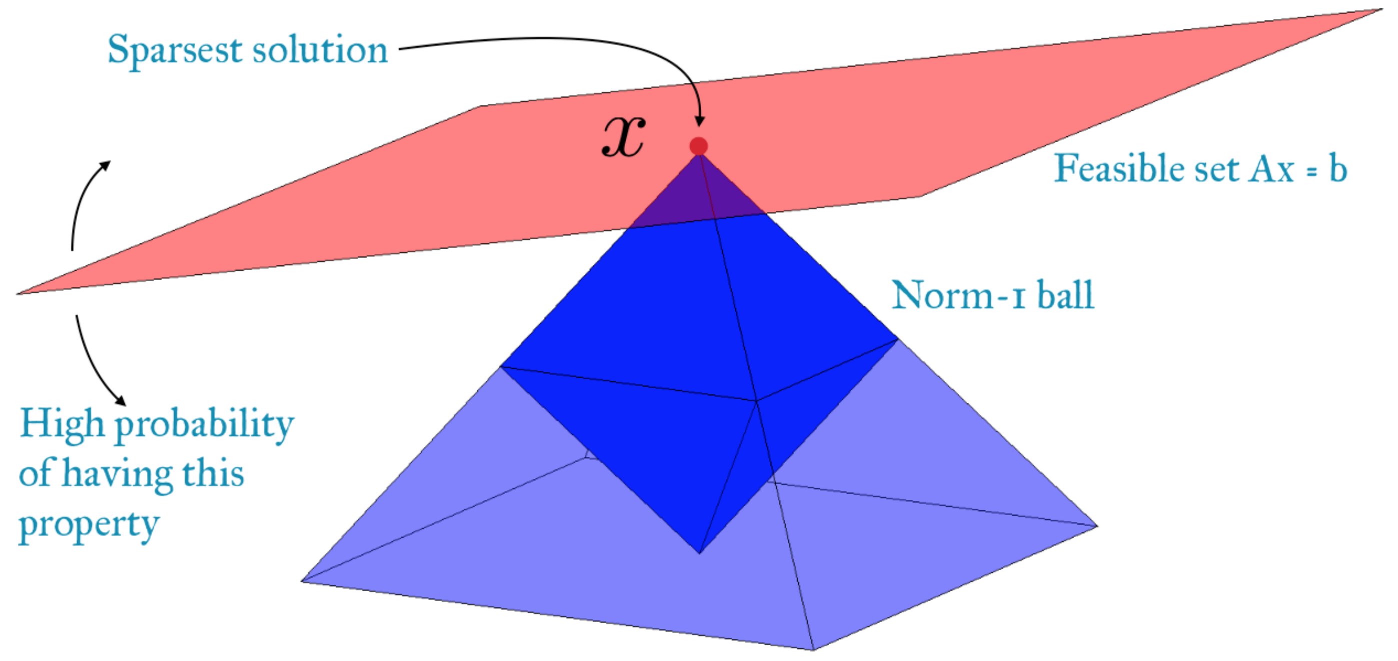
$$\boxed{\left.\begin{array}{rrcl} \min & \sum_{j\le n} & s_j & \\ \forall j \le n & -s_j \le & x_j & \le s_j \\ & Ax & = & b \end{array}\right\}}$$

- **Empirical observation: can often find optimum**

  Too often for this to be a coincidence

- **Theoretical justification by Candès, Tao, Donoho**
  "Mathematics of sparsity", "Compressed sensing"

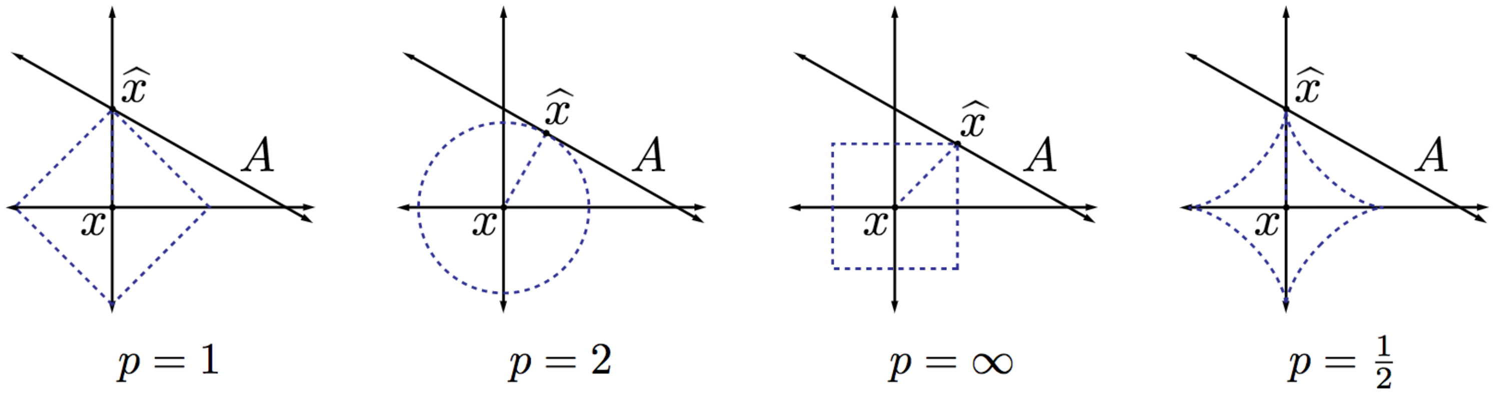
Graphical intuition 1



- **Wouldn't work with ℓ_2, ℓ_∞ norms**
  $Ax = b$ ***flat at poles*** — **"zero probability of sparse solution"**

Warning: *this is not a proof, and there are cases not explained by this drawing* [Candès 2014]

Graphical intuition 2



- $\hat{x}$ **such that** $A\hat{x} = b$ **approximates** x **in** ℓ_p **norms**
- $p = 1$ **only convex case zeroing some components**

From [Davenport et al., 2012]; again, this is not a proof!

Not for the faint-hearted

1. Hand, Voroninski:

   `arxiv.org/pdf/1611.03935v1.pdf`
2. Candès and Tao:

   `statweb.stanford.edu/~candes/papers/DecodingLP.pdf`
3. Candès:

   `statweb.stanford.edu/~candes/papers/ICM2014.pdf`
4. Davenport *et al.*:

   `statweb.stanford.edu/~markad/publications/ddek-chapter1-2011.pdf`
5. Lustig *et al.*:

   `people.eecs.berkeley.edu/~mlustig/CS/CSMRI.pdf`
6. *and many more* (look for "compressed sensing")