15.094J/1.142J Robust Modeling, Optimization and Computation	**Homework 1**	Exercise 1.1 **By: Hanzhang Qin**

(a) For simplicity, we only consider the core problem of finding the robust counterpart:

$$\max_{\mathbf{c}\in\mathcal{U}} \sum_{(i,j)\in E} c_{ij}x_{ij} \le t$$

since we only have uncertainties in the objective function, and we can reformulate the optimization problem as

$$\begin{aligned} &\min\ t \\ &\text{s.t.} \sum_{(i,j)\in E} c_{ij}x_{ij} \le t \\ &\qquad [\text{Flow constraints}] \end{aligned}$$

Note I omit the flow constraints since they are not relevant to the process of developing the robust counterparts. Note we can reduce the uncertainty on $\mathbf{c}$ to γ, by writing $c = \mu + \gamma\sigma$ and thus our problem is equivalent to

$$\max_{\|\gamma\|\le\Gamma} \sum_{(i,j)\in E} x_{ij}\delta_{ij}\gamma_{ij} \le t - \sum_{(i,j)\in E} x_{ij}\mu_{ij} \qquad (*)$$

(i) Box uncertainty: This case is trivial because we have

$$\max_{\|\gamma\|_\infty\le\Gamma} \sum_{(i,j)\in E} x_{ij}\delta_{ij}\gamma_{ij} = \Gamma \sum_{(i,j)\in E} x_{ij}\delta_{ij}$$

since all x_{ij}, δ_{ij} are nonnegative.

(ii) Polyhedral uncertainty: We consider the matrix formulation of problem (*) with the l_1 uncertainty set: (note we treat γ as a n^2 long vector, avoiding the vec notation)

$$\begin{aligned} \max\ & \underbrace{[(x\delta)^T\ 0^T]}_{f^T} \begin{bmatrix} \gamma \\ \tilde{\gamma} \end{bmatrix} \\ \text{s.t.}\ & \underbrace{\begin{bmatrix} 0 & e^T \\ I & -I \\ -I & -I \end{bmatrix}}_{D} \begin{bmatrix} \gamma \\ \tilde{\gamma} \end{bmatrix} \le \underbrace{\begin{bmatrix} \Gamma \\ 0 \\ 0 \end{bmatrix}}_{d}. \end{aligned}$$

Then from slides we know the robust counterpart of (*) is

$$\begin{aligned} & d^T y \le t - \sum_{(i,j)\in E} x_{ij}\mu_{ij}, \\ & D^T y = f, \\ & y \ge 0. \end{aligned}$$

(iii) Ellipsoidal Uncertainty: We know from slides the l_2 robust counterpart of (*) is a second order cone problem as follows.

$$\sum_{(i,j)\in E} x_{ij}\mu_{ij} + \Gamma\|\delta x\|_2 \le t.$$

(b) We conduct the numerical experiments in Julia. We report the objective values under simulated Gaussian costs in Table 1, by ranging n within $\{10, 25, 50, 75, 100\}$ and Γ within $\{0, 1e^{-4}, 1e^{-1}, 1e^{1}, 1e^{3}\}$.

Instance/n	10	25	50	75	100
Nominal	9.39067	2.22615	1.13683	0.909956	0.052574
$l_\infty(\Gamma = 1e^{-4})$	9.39067	2.22615	1.13683	0.909956	0.052574
$l_1(\Gamma = 1e^{-4})$	9.39067	2.22615	1.13683	0.909956	0.052574
$l_2(\Gamma = 1e^{-4})$	9.39067	2.22615	1.13683	0.909956	0.052574
$l_\infty(\Gamma = 1e^{-1})$	9.39067	1.48855	1.13683	0.909956	0.052574
$l_1(\Gamma = 1e^{-1})$	9.39067	1.69325	1.13683	0.909956	0.052574
$l_2(\Gamma = 1e^{-1})$	9.39067	1.61247	1.13683	0.89898	0.052574
$l_\infty(\Gamma = 1e^{1})$	10.6399	1.48855	1.63361	1.43819	0.419424
$l_1(\Gamma = 1e^{1})$	9.20335	1.8294	1.41089	1.01207	0.385019
$l_2(\Gamma = 1e^{1})$	9.54801	1.62139	1.47901	0.978531	0.40741
$l_\infty(\Gamma = 1e^{3})$	10.6399	2.6358	5.80325	8.1277	0.419424
$l_1(\Gamma = 1e^{3})$	10.24	3.91471	3.95935	4.45947	3.06828
$l_2(\Gamma = 1e^{3})$	11.3405	5.3878	6.74021	10.0393	6.15165

Table 1: Computational results for Exercise 1.1 (simulated objective values)

I observe that typically the l_∞ set gives the most conservative solution (i.e., biggest objective values) while the l_1 set has the smallest support (l_2 in between). Moreover, for a fixed n, as Γ increases, the solution tend to be more conservative (the objective values increases). As n increases, typically the objective values become smaller since we are testing on a fully connected graph.

(a) Using the knowledge of robust counterpart under general norms from the slides, we know

$$\max_{\|a-\hat{a}\|\leq\epsilon} a^T x \leq b \Leftrightarrow \hat{a}^T x + \epsilon\|x\|_1 \leq b.$$

To make the system entirely linear, we just need to introduce auxiliary variables so that the system becomes

$$\begin{aligned}
&\hat{a}^T x + \epsilon \sum_i z_i \leq b,\\
&z_i \geq x_i, \forall\ i,\\
&z_i \geq -x_i, \forall\ i.
\end{aligned}$$

(b) Obviously constraints $z_i \geq x_i$ and $z_i \geq -x_i$ are sparse, but the constraint $\hat{a}^T x + \epsilon \sum_i z_i \leq b$ is dense since this constraint involves all the variables (number of z_i is huge).

(c) Similar to (a), we obtain the robust counterpart as

$$\begin{aligned}
&\hat{a}^T x + \epsilon \sum_i z \leq b,\\
&z \geq x_i, \forall\ i,\\
&z \geq -x_i, \forall\ i.
\end{aligned}$$

Note this time the system preserves the sparsity structure of the primal constraint, since in this case we only need to add one auxiliary variable z.

(d) This example is a simple illustration of how to choose uncertainty sets smartly. When we want to preserve some nice structures of the primal deterministic system, we really need to be very careful to construct the uncertainty sets. After all, l_1 constraints and l_∞ constraints might not differ much in theory, but might differ much in practice!

Pset1_JuliaCode

February 22, 2017

In [2]:
```
using JuMP, JuMPeR, Gurobi, Distributions
```

In [3]:
```
# Nominal Problem
function NominalProblem(n,μ,δ)
    NetworkModel = Model(solver=GurobiSolver(OutputFlag=0))
    capacity = (ones(n,n)-eye(n,n))*0.5
    capacity[1,n] = 0
    capacity[n,1] = 0
    @variable(NetworkModel, flow[i=1:n,j=1:n]>=0)
    @constraint(NetworkModel, flow .<= capacity)
    @constraint(NetworkModel, sum(flow[1,i] for i=1:n)==1)
    @constraint(NetworkModel, sum(flow[i,n] for i=1:n)==1)
    for j = 2:n-1
        @constraint(NetworkModel, sum(flow[i,j] for i=1:n) == sum(flow[j,k] for k=1:n)  )
    end
    @objective(NetworkModel,Min, sum{flow[i,j]*μ[i,j], i=1:n,j=1:n} );
    solve(NetworkModel)
    return getvalue(flow)
end
```

Out[3]:
```
NominalProblem (generic function with 1 method)
```

In [4]:
```
# Robust Problem
function RobustProblem(n,μ,δ,Γ,norm_type)
    NetworkModel_robust = RobustModel(solver=GurobiSolver(OutputFlag=0))
    capacity = (ones(n,n)-eye(n,n))*0.5
    capacity[1,n] = 0
    capacity[n,1] = 0
    @variable(NetworkModel_robust, flow[i=1:n,j=1:n]>=0)
    @uncertain(NetworkModel_robust, cost[i=1:n,j=1:n])
    @uncertain(NetworkModel_robust, r[i=1:n,j=1:n])
    @variable(NetworkModel_robust, obj)

    for i = 1:n
        for j = 1:n
        @constraint(NetworkModel_robust, cost[i,j] == μ[i,j]+r[i,j]*δ[i,j])
        end
    end

    @constraint(NetworkModel_robust, norm(r,norm_type) <= Γ)

    @constraint(NetworkModel_robust, flow .<= capacity)
    @constraint(NetworkModel_robust, sum(flow[1,i] for i=1:n) == 1)
    @constraint(NetworkModel_robust, sum(flow[i,n] for i=1:n) == 1)
```

```
            for j = 2:n-1
                @constraint(NetworkModel_robust, sum(flow[i,j] for i=1:n) == sum(flow[j,k] for k=1:n))
            end
            @constraint(NetworkModel_robust, sum{cost[i,j]*flow[i,j],i=1:n,j=1:n} <= obj )
            @objective(NetworkModel_robust, Min, obj)
            solve(NetworkModel_robust)
            return getvalue(flow)
        end
```

Out[4]: RobustProblem (generic function with 1 method)

```
In [8]: function Evaluation()
            n_type = [10,25,50,75,100]
            Γ_type = [1e-4,1e-1,1e1,1e4]
            report_data = zeros(13,5)
            for n_idx = 1:5
                n = n_type[n_idx]
                count = 2
                μ = rand(Uniform(0,10),n,n)
                δ = zeros(n,n)
                for i = 1:n
                    for j = 1:n
                        δ[i,j] = rand(Uniform(0,μ[i,j]))
                    end
                end
                cost_sim = zeros(n,n)
                        for i = 1:n
                            for j = 1:n
                                cost_sim[i,j] = rand(Normal(μ[i,j],δ[i,j]))
                            end
                        end
                sol_0 = NominalProblem(n,μ,δ)
                report_data[1,n_idx] = sum(sum(cost_sim.*sol_0))
                for Γ in Γ_type
                    print("n=",n," Γ=",Γ,"\n")
                    sol_inf = RobustProblem(n,μ,δ,Γ,Inf)
                    sol_1 = RobustProblem(n,μ,δ,Γ,1)
                    sol_2 = RobustProblem(n,μ,δ,Γ,2)
                    report_data[count,n_idx] = sum(sum(cost_sim.*sol_inf))
                    report_data[count+1,n_idx] = sum(sum(cost_sim.*sol_1))
                    report_data[count+2,n_idx] = sum(sum(cost_sim.*sol_2))

                    count = count + 3
                end
            end
            return report_data
        end
```

```
WARNING: Method definition Evaluation() in module Main at In[5]:2 overwritten at In[8]:2.
```

Out[8]: Evaluation (generic function with 1 method)

In [ ]: