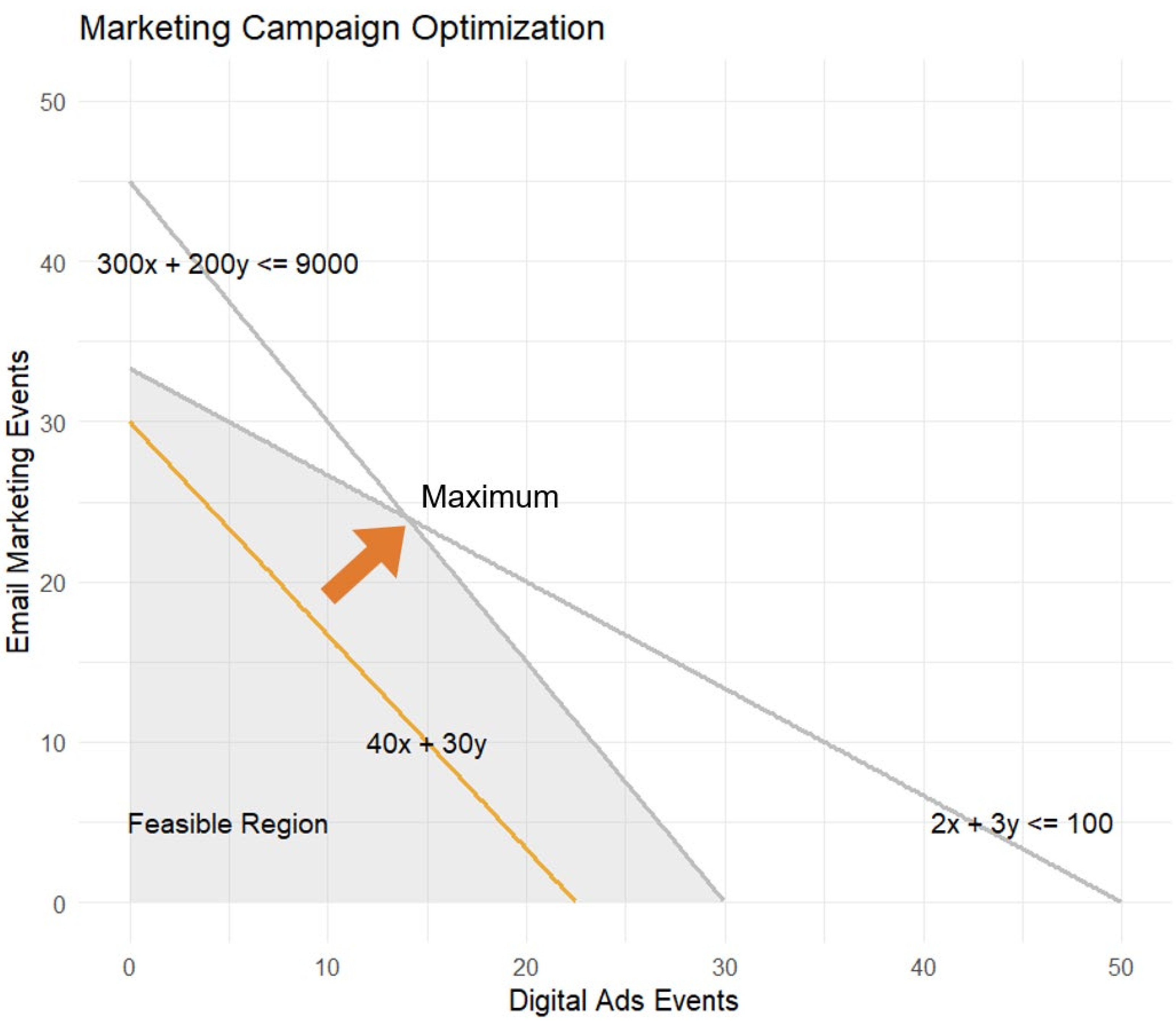
Prescriptive Analytics

Business Intelligence



Terry College of Business
UNIVERSITY OF GEORGIA

Exercise

A power plant forecasts a need for 2000 mWh for the next hour. The EPA requires that the emission of pollutants is below 900 kg per hour. The plant has several options, and importing power is limited to 200 mWh. It wants to minimize costs.

Method	Pollution	Cost/mWh $
High-sulfur fuel	4.5 kg/mWh	3.50
Low-sulfur fuel	0.54 kg/mWh	5.00
Stack filters	90% reduction	0.80
Import power	No local pollutants	4.00

Exercise

- How many variables?
  - High-sulfur fuel
  - Low-sulfur fuel
  - High-sulfur fuel with stack filters
  - Low-sulfur fuel with stack filters
  - Imported power

Exercise

What are the pollutants released and cost for each of the variables?

Variables	Method	Pollution (kg/mWh)	Cost/mWh $
x1	High-sulfur fuel	4.500	3.50
x2	Low-sulfur fuel	0.540	5.00
x3	High-sulfur fuel with stack filters	0.450	4.30
x4	Low-sulfur fuel with stack filters	0.054	5.80
x5	Import power	0.000	4.00

Minimize: 3.5*x1 + 5*x2 + 4.3*x3 + 5.8*x4 + 4*x5

Exercise

What are the constraints?

- At least 2000 mWh
  $x1 + x2 + x3 + x4 + x5 \geq 2000$
- Pollutants less than 900 kg per hour
  $4.5*x1 + .54*x2 + .45*x3 + .054*x4 \leq 900$
- Imported power is limited to 200 mWh
  $x5 \leq 200$

Exercise code

Parameter Specification

```
library("lpSolveAPI")
# create a new model with
# 0 constraints and 5 decision variables
lpmodel2 <- make.lp(0, 5)
# Code for reproducibility
pollution_limit <- 900
required_mWh <-  2000
max_import <- 200
# Costs
stack_filter_cost = .80
c1 <- 3.5 # High-sulfur fuey
c2 <- 5.0 # Low-sulfur fuel
c3 <- c1 + stack_filter_cost
c4 <- c2 + stack_filter_cost
c5 <- 4.0
# Pollution
stack_filter_reduction = .90
p1 <- 4.5
p2 <- .54
p3 <- p1*(1 - stack_filter_reduction)
p4 <- p2*(1 - stack_filter_reduction)
p5 <- 0
```

Problem Formulation and Solution

```
# minimize costs
set.objfn(lpmodel2, c(c1,c2,c3,c4,c5))
# demand constraint
add.constraint(lpmodel2, c(1,1,1,1,1), ">=", required_mWh)
# supply constraints
add.constraint(lpmodel2, c(p1,p2,p3,p4,p5), "<=", pollution_limit)
add.constraint(lpmodel2, c(0,0,0,0,1), "<=", max_import)
#set objective direction and hide the output
invisible(lp.control(lpmodel2, sense = 'min'))
print(lpmodel2)
solve(lpmodel2)
# what is the minimum cost?
get.objective(lpmodel2)
# what are the values of the decision variables?
get.variables(lpmodel2)
# what are the shadow prices?
get.dual.solution(lpmodel2)
```

Exercise Results - Pollutants ≤ 900

Cost = 8522.22

Variables	Method	Values
x1	High-sulfur fuel	22.22
x2	Low-sulfur fuel	0
x3	High-sulfur fuel with stack filters	1777.78
x4	Low-sulfur fuel with stack filters	0
x5	Import power	200

Modified Example - Alternative Constraint

- What happens if the EPA changes the limit to 700 kg per hour?
  - Set limit = 700 and rerun
- Cost function increases to $8,957

Variables	Method	Values
x1	High-sulfur fuel	0
x2	Low-sulfur fuel	0
x3	High-sulfur fuel with stack filters	1522.22
x4	Low-sulfur fuel with stack filters	277.78
x5	Import power	200

- Greater use of stack filters (x3 and x4)

Transportation problem

- Warehouses in Detroit, Pittsburgh, and Buffalo have 250, 130, and 235 tons of paper, respectively
- Print shops in Boston, New York, Chicago, and Indianapolis, respectively, ordered 75, 230, ,240 and 70 tons of paper
- Minimize the cost of shipment
  - How much to ship from each warehouse to each print shop?
  - 12 decision variables

Cost of shipment from\to	Boston (BS)	New York (NY)	Chicago (DH)	Indianapolis (IN)
Detroit (DT)	15	20	16	21
Pittsburgh (PT)	25	13	5	11
Buffalo (BF)	15	15	7	17

Problem Formulation

```
# create a new model with 0 constraints and 12 decision variables
library("lpSolveAPI")
supplyConstraints <-  3
demandConstraints <-  4
lpmodel <- make.lp(0, supplyConstraints*demandConstraints)
# minimize costs
# The objective function is sum of multiplications of transported tons and their costs
# 15*DTtoBS + 20*DTtoNY + …
set.objfn(lpmodel, c(15,20,16,21,
                     25,13,5,11,
                     15,15,7,17))
# supply constraints
add.constraint(lpmodel, c(1,1,1,1,
                          0,0,0,0,
                          0,0,0,0), "<=", 250)
add.constraint(lpmodel, c(0,0,0,0,
                          1,1,1,1,
                          0,0,0,0), "<=", 130)
add.constraint(lpmodel, c(0,0,0,0,
                          0,0,0,0,
                          1,1,1,1), "<=", 235)
# demand constraints
add.constraint(lpmodel, c(1,0,0,0,
                          1,0,0,0,
                          1,0,0,0), ">=", 75)
add.constraint(lpmodel, c(0,1,0,0,
                          0,1,0,0,
                          0,1,0,0), ">=", 230)
add.constraint(lpmodel, c(0,0,1,0,
                          0,0,1,0,
                          0,0,1,0), ">=", 240)
add.constraint(lpmodel, c(0,0,0,1,
                          0,0,0,1,
                          0,0,0,1), ">=", 70)
#set objective direction and hide the output
invisible(lp.control(lpmodel, sense = 'min'))
```

Solution

```
solve(lpmodel)
[1] 0
# what is the minimum cost?
get.objective(lpmodel)
[1] 7780
# what are the values of the decision variables?
get.variables(lpmodel)
 [1]  75 175   0   0   0  55   5  70   0   0 235   0
# Turn vector into matrix and transpose so origin is row
solution <-
t(matrix(get.variables(lpmodel),demandConstraints,supplyConstraints))
RowNames <- c("Detroit", "Pittsburgh", "Buffalo")
ColNames <- c("Boston","NewYork","Chicago","Indianapolis")
dimnames(solution) <-  list(RowNames, ColNames)
solution
           Boston NewYork Chicago Indianapolis
Detroit        75     175       0            0
Pittsburgh      0      55       5           70
Buffalo         0       0     235            0
```

Legend

R script input
Console output

Tons shipped from\to	Boston	New York	Chicago	Indianapolis	Supply
Detroit	75	175	0	0	250
Pittsburgh	0	55	5	70	130
Buffalo	0	0	235	0	235
Demand	75	230	240	70	615

Shadow prices

```
get.dual.solution(lpmodel)
[1]  1  0 -7 -5 15 20 12 18  0  0  4  3 17  0  0  0  5  0  0  4
shadowPrices <-
get.dual.solution(lpmodel)[2:(supplyConstraints+demandConstraints+1)]
shadowPrices
# shadowPrices for supply
shadowPrices[1:(supplyConstraints)]
# shadowPrices for demand
shadowPrices[(supplyConstraints+1):(supplyConstraints+demandConstraints)]
```

Constraint	Detroit	Pittsburgh	Buffalo	Boston	New York	Chicago	Indianapolis
Shadow price	0	-7	-5	15	20	12	18

Increasing the capacity of Pittsburgh will reduce costs by $7

Reduced costs

Reduced cost, or opportunity cost, is the amount by which an objective function coefficient would have to change (increase for maximization problem, decrease for minimization problem) before it would be possible for the corresponding variable to have a positive value in the optimal solution.

```
get.dual.solution(lpmodel)
 [1]   1   0 -7 -5 15 20 12 18  0  0  4  3 17  0  0  0  5  0  0  4
r <-
get.dual.solution(lpmodel)[(supplyConstraints+demandConstraints+
2):(supplyConstraints+demandConstraints+ 1
+supplyConstraints*demandConstraints)]
reducedCosts <-
t(matrix(r,demandConstraints,supplyConstraints))
dimnames(reducedCosts) <-  list(RowNames, ColNames)
reducedCosts
           Boston NewYork Chicago Indianapolis
Detroit         0       0       4            3
Pittsburgh     17       0       0            0
Buffalo         5       0       0            4
```

Tons shipped from\to	Boston	New York	Chicago	Indianapolis
Detroit	75	175	0	0
Pittsburgh	0	55	5	70
Buffalo	0	0	235	0

Reduced cost from\to	Boston	New York	Chicago	Indianapolis
Detroit	0	0	4	3
Pittsburgh	17	0	0	0
Buffalo	5	0	0	4

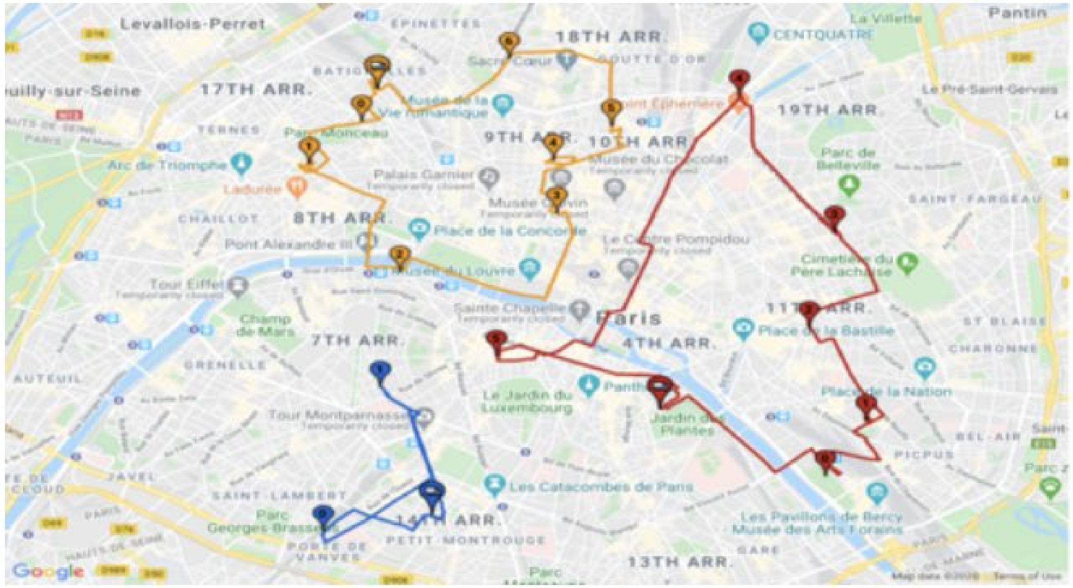
Traveling Salesperson Problem (TSP)

Traveling Salesperson Problem (TSP)

- Find the shortest path that visits each city in a given list exactly once and then returns to the starting city
  - Vehicle routing
  - Computer wiring

Algorithms - TSP

https://www.youtube.com/watch?v=k2AqGongii0&t=687s

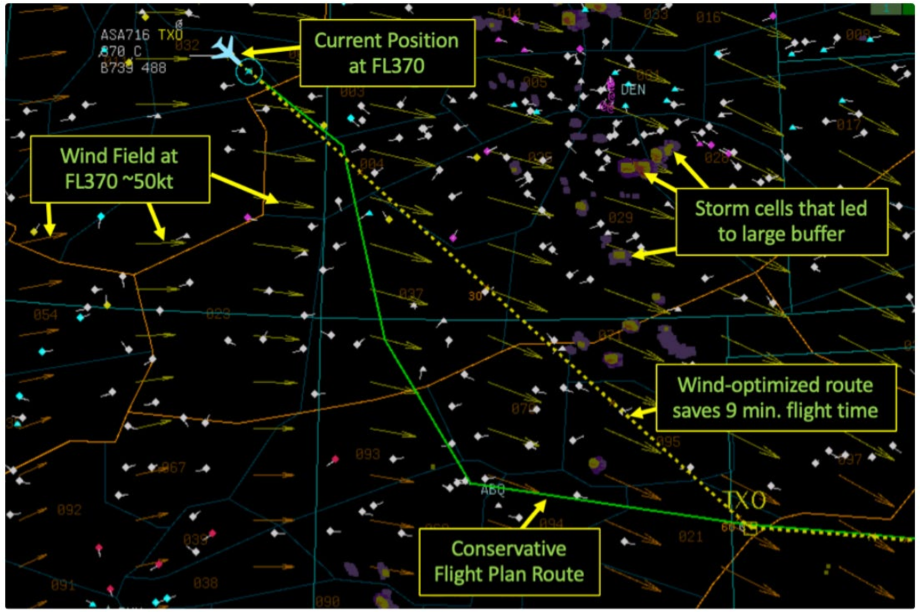


Route Optimization API

The Route Optimization API assigns tasks and routes to a vehicle fleet, optimizing against the objectives and constraints that you supply for your transportation goals.

Wind – One of Five Influences in Route Optimization

AUGUST 7, 2020



Conclusion

- Linear programming can be used to solve problems that are sufficiently well-defined to be described as a set of linear equations.

- There are extensions for integer solutions and non-linear situations.

- The traveling salesperson algorithm computes a solution that is close to the optimum.