Exploiting Sparsity

Shuvomoy Das Gupta

June 21, 2019

This tutorial describes how to make an optimization model more efficient by exploiting sparsity.

1 Introduction

1.1 What is a sparse data structure?

A sparse data structure is one that has a lot of zeros in it. If a matrix has many more zeros than nonzeros, then it is a sparse matrix.

1.2 Why do we need to exploit sparsity?

- Sparsity in the input data increases with the dimension.
- Exploiting sparsity
  - keeps the data size small
  - saves memory
  - reduces the running time
  - improves the efficiency of the model

1.3 How to exploit sparsity in Julia?

- Define `struct` and create a `dictionary` or an `array` of it.

2 A test example: Transportation Problem

Consider a transportation problem which is going to be our test example:

- **Problem setup:** Some products have to transported from origin cities to destination cities
- **Objective:** Minimize the total cost of shipment over all **relevant** routes

- **Decision Variables:** Find the optimum amount of every product to be shipped from one city to another

- **Constraints:**

  - How much of a product a city can supply to other cities is fixed.
  - The amount of any product demanded by a city is also fixed.
  - The total amount of products shipped between every pair of different cities can not exceed a given limit.

Suppose there are ten cities and three products in our problem.

```
cities =
[
:BANGKOK; :LONDON; :PARIS; :SINGAPORE;  :NEWYORK;  :ISTANBUL;  :DUBAI;  :KUALALUMPUR;
    :HONGKONG;  :BARCELONA
]

products   =
[
:smartphone; :tablet; :laptop
]

3-element Array{Symbol,1}:
 :smartphone
 :tablet
 :laptop
```

3 Defining structs

If we do not exploit sparsity, then number of ways we can ship the products from one city to other will be $3 \times ({}^{10}P_2 + 10) = 300$.

Clearly many of them will be redundant, because of reasons like

- a product might not be needed by a city
- a product might not be produced by a city etc.

We just need to consider *relevant route*s, where a product can be shipped from one production city to the other demand city. So a *relevant route* can be defined by 3 features:

- a product
- a city that produces that product
- a city that demands that product

So, we define an `struct Route` as follows:

```
struct Route
    p::Symbol # p stands for product
    o::Symbol # o stands for origin
    d::Symbol # d stands for destination
end
```

Here the datatype `Symbol` is a special type of immutable string. Then we create an array of only relevant routes.

```
routesExample =
[
  Route(:smartphone,:BANGKOK,:SINGAPORE);
  Route(:smartphone,:BANGKOK,:NEWYORK);
  Route(:smartphone,:BANGKOK,:ISTANBUL);
  Route(:smartphone,:BANGKOK,:DUBAI);
]
```

```
4-element Array{Main.WeaveSandBox26.Route,1}:
 Main.WeaveSandBox26.Route(:smartphone, :BANGKOK, :SINGAPORE)
 Main.WeaveSandBox26.Route(:smartphone, :BANGKOK, :NEWYORK)
 Main.WeaveSandBox26.Route(:smartphone, :BANGKOK, :ISTANBUL)
 Main.WeaveSandBox26.Route(:smartphone, :BANGKOK, :DUBAI)
```

If we want to access ith element of the array by typing in `routes[i]`. When we want to access the product name associated with the ith element of the array, we can do so by typing `routes[i].p`.

```
routesExample[2] # Will give the second route
```

```
Main.WeaveSandBox26.Route(:smartphone, :BANGKOK, :NEWYORK)
```

```
routesExample[4].d # Will give the demand city of the 4th route
```

```
:DUBAI
```

4 Creating new arrays efficiently from existing arrays

Often we need to create new arrays, where the elements of them are extracted from some already existing array conditionally. Consider the immutable type `Supply`

```
struct Supply
    p::Symbol # p stands for product name
    o::Symbol # o stands for the origin city
end
```

We want to create an array `suppliesExample`, that contains all relevant product-city pairs, where the particular product is produced in that city. Clearly we can construct this array by plucking each product and corresponding city producing it from `routesExample`. This is how we do it efficiently:

- Create an empty array of type `Supply`
- Add elements to this array by
  - selecting the product and origin from the elements of `routes`

- *push*ing them one by one in `supplies`

```
suppliesExample = Supply[] # Creates a 0 element array of immutable type Supply
for r in routesExample # For every element of the route route
    push!(suppliesExample, Supply(r.p, r.o)) # pick the product and origin city and push
    it in supplies
end
```

```
suppliesExample
```

```
4-element Array{Main.WeaveSandBox26.Supply,1}:
 Main.WeaveSandBox26.Supply(:smartphone, :BANGKOK)
 Main.WeaveSandBox26.Supply(:smartphone, :BANGKOK)
 Main.WeaveSandBox26.Supply(:smartphone, :BANGKOK)
 Main.WeaveSandBox26.Supply(:smartphone, :BANGKOK)
```

5 Dictionary

5.1 What is a dictionary?

A dictionary is a data type which can be useful in exploiting sparsity.

5.2 Why is it needed?

Often we might be interested to index a variable by a composite data type, rather than a number.

For example, for the transportation problem in consideration, it would be more convenient to index the decision variables in the routes that are present. Let

$$R = \{(p, o, d) \in P \times C \times C : \text{product } p \text{ has to be transported from city } o \text{ to city } d\}$$

be the set of all the routes that are relevant for the problem. So, we can define our decision variables as below:

$$\forall (p, o, d) \in R \left(x_{(p,o,d)} = \text{the amount of a product } p \text{ that is transported from city } o \text{ city d} \right)$$

From a data structure point of view, \$ \left(x*{(p,o,d)}\right)*{(p,o,d) \in R}\$ is a dictionary which

- takes $(p, o, d) \in R$ as its **key** and
- has the **value** the optimum amount of the product p to be shipped from city o to city d.

5.3 Efficiently Constructing Dictionary of structs

Suppose we want to create a dictionary called `costRoutes`. Every element of the dictionary `costRoutes` contains the value of shipping cost along a particular route belonging to the array `routesExample`. So,

- the **key** to an element belonging to the dictionary is a specific route belonging to the array `routesExample`
- the **value** is the cost for that shipment.

Suppose the values of the costs are stored in an array named `costCofExample`.

```
costCofExample = [120; 205; 310; 45.0]
```

```
4-element Array{Float64,1}:
 120.0
 205.0
 310.0
  45.0
```

We create the dictionary `costRoutes` similar to an array:

- we create an empty dictionary, and then
- use the command

`setindex!(name_of_dictionary, value, key)` or `name_of_dictionary[key]=value` to add new elements in the dictionary one by one

```
costRoutesExample=Dict{Route, Float64}()# Create an empty dictionary
```

```
Dict{Main.WeaveSandBox26.Route,Float64} with 0 entries
```

where the key is `Route` and the value is `Float64`

```
for i in 1:length(routesExample)
   costRoutesExample[routesExample[i]]=costCofExample[i] # routesExample[i] is the key,
    and costCofExample[i] is the value
end
```

```
costRoutesExample
```

```
Dict{Main.WeaveSandBox26.Route,Float64} with 4 entries:
  Route(:smartphone, :BANGKOK, :DUBAI)     => 45.0
  Route(:smartphone, :BANGKOK, :NEWYORK)   => 205.0
  Route(:smartphone, :BANGKOK, :SINGAPORE) => 120.0
  Route(:smartphone, :BANGKOK, :ISTANBUL)  => 310.0
```

After the dictionary is initialized, we can access the cost associated with some route `routes[i]` by typing in `costRoutes[routes[i]]`

```
costRoutesExample[routesExample[4]]
```

```
45.0
```

Or we can input the description of the route itself:

```
routesExample[3]
```

```
Main.WeaveSandBox26.Route(:smartphone, :BANGKOK, :ISTANBUL)
```

```
costRoutesExample[Route(:smartphone,:BANGKOK,:ISTANBUL)]
```

```
310.0
```

6 Mathematical representation of the transportation problem

The problem is a classic transportation problem. We will consider the sparse representation of the problem. Let

$$C = \text{Set of all cities}$$

$$P = \text{Set of all products}$$

$$R = \{(p, o, d) \in P \times C \times C : \text{product } p \text{ has to be transported from city } o \text{ to city } d\}$$

$$\forall (p, o, d) \in R \quad \left(c_{(p,o,d)} = \text{cost of transporting some product } p \text{ from city } o \text{ to city } d\right)$$

$$\forall p \in P \quad (O_p = \text{Set of all origin cities where a product } p \text{ is produced})$$

$$\forall p \in P \quad (D_p = \text{Set of all destination cities where a product } p \text{has to be delivered})$$

$$\forall p \in P\ \forall o \in O_p \quad \left(s_{(p,o)} = \text{the total amount of the product } p \text{ that can be supplied by city } o\right)$$

$$\forall p \in P\ \forall d \in D_p \quad \left(d_{(p,d)} = \text{the total amount of the product } p \text{ that is demanded by city } d\right)$$

Total amount of shipped product between each pair of cities cannot exceed γ #' The decision variable for this problem is

$$\forall (p, o, d) \in R \quad \left(x_{(p,o,d)} = \text{the amount of a product } p \text{ that is trasported from city } o \text{ city d}\right)$$

The optimization problem can be described as below:

$$\text{minimize} \quad \sum_{(p,o,d)\in R} c_{(p,o,d)} x_{(p,o,d)}$$

subject to

$$\forall p \in P \ \forall o \in O_p \quad \left(\sum_{d \in D_p} x_{(p,o,d)} = s_{(p,o)} \right)$$

: The amount of any product a city can supply to other cities is fixed

$$\forall p \in P \ \forall d \in D_p \quad \left(\sum_{o \in O_p} x_{(p,o,d)} = d_{(p,d)} \right)$$

: The amount of any product demanded by a city is also fixed.

$$\forall o \in C \ \forall d \in C \setminus \{o\} \quad \left(\sum_{(p,\bar{o},\bar{d})\in R \,:\, o=\bar{o} \wedge d=\bar{d}} x_{(p,\bar{o},\bar{d})} \leq \gamma \right)$$

:The total amount of products shipped between every pair of different cities can not exceed

7 Mapping of the mathematical symbols to JuMP

In the data file, the symbols in the model above are mapped as follows:

Mathematical Symbol	In the code	
C	`cities`	`cities` is an array
P	`products`	`products` is an array
R	`routes`	`routes` is an array of immutable
$(O_p)_{p\in P}$	`orig`	`orig` is
$(D_p)_{p\in P}$	`dest`	`dest` is
$((s_{(p,o)})_{o\in O_p})_{p\in P}$	`suppliedAmount`	`suppliedAmount` is
$((d_{(p,d)})_{d\in D_p})_{p\in P}$	`demandedAmount`	`demandedAmount` is a dictionary with key `Customer` and va
$(x_{(p,o,d)})_{(p,o,d)\in R}$	`trans`	`trans` is a dictionary and the variable in
$(c_{(p,o,d)})_{(p,o,d)\in R}$	`costRoutes`	`costRoutes` is
γ	`capacity`	It is of ty

8 Problem Data

```
cities =
[
:BANGKOK; :LONDON; :PARIS; :SINGAPORE;  :NEWYORK;  :ISTANBUL;  :DUBAI;  :KUALALUMPUR;
    :HONGKONG;  :BARCELONA
]

products   =
[
:smartphone; :tablet; :laptop
]

capacity    = 700

struct Route
```

```julia
    p::Symbol # p stands for product
    o::Symbol # o stands for origin
    d::Symbol # d stands for destination
end

routes =
[
  Route(:smartphone,:BANGKOK,:SINGAPORE);
  Route(:smartphone,:BANGKOK,:NEWYORK);
  Route(:smartphone,:BANGKOK,:ISTANBUL);
  Route(:smartphone,:BANGKOK,:DUBAI);
  Route(:smartphone,:BANGKOK,:KUALALUMPUR);
  Route(:smartphone,:BANGKOK,:HONGKONG);
  Route(:smartphone,:BANGKOK,:BARCELONA);
  Route(:smartphone,:LONDON,:SINGAPORE);
  Route(:smartphone,:LONDON,:NEWYORK);
  Route(:smartphone,:LONDON,:ISTANBUL);
  Route(:smartphone,:LONDON,:DUBAI);
  Route(:smartphone,:LONDON,:KUALALUMPUR);
  Route(:smartphone,:LONDON,:HONGKONG);
  Route(:smartphone,:LONDON,:BARCELONA);
  Route(:smartphone,:PARIS,:SINGAPORE);
  Route(:smartphone,:PARIS,:NEWYORK);
  Route(:smartphone,:PARIS,:ISTANBUL);
  Route(:smartphone,:PARIS,:DUBAI);
  Route(:smartphone,:PARIS,:KUALALUMPUR);
  Route(:smartphone,:PARIS,:HONGKONG);
  Route(:smartphone,:PARIS,:BARCELONA);

  Route(:tablet,:BANGKOK,:SINGAPORE);
  Route(:tablet,:BANGKOK,:NEWYORK);
  Route(:tablet,:BANGKOK,:ISTANBUL);
  Route(:tablet,:BANGKOK,:DUBAI);
  Route(:tablet,:BANGKOK,:KUALALUMPUR);
  Route(:tablet,:BANGKOK,:HONGKONG);
  Route(:tablet,:BANGKOK,:BARCELONA);

  Route(:tablet,:LONDON,:SINGAPORE);
  Route(:tablet,:LONDON,:NEWYORK);
  Route(:tablet,:LONDON,:ISTANBUL);
  Route(:tablet,:LONDON,:DUBAI);
  Route(:tablet,:LONDON,:KUALALUMPUR);
  Route(:tablet,:LONDON,:HONGKONG);
  Route(:tablet,:LONDON,:BARCELONA);

  Route(:tablet,:PARIS,:SINGAPORE);
  Route(:tablet,:PARIS,:NEWYORK);
  Route(:tablet,:PARIS,:ISTANBUL);
  Route(:tablet,:PARIS,:DUBAI);
  Route(:tablet,:PARIS,:KUALALUMPUR);
  Route(:tablet,:PARIS,:HONGKONG);
  Route(:tablet,:PARIS,:BARCELONA);

  Route(:laptop,:BANGKOK,:SINGAPORE);
  Route(:laptop,:BANGKOK,:NEWYORK);
  Route(:laptop,:BANGKOK,:ISTANBUL);
  Route(:laptop,:BANGKOK,:DUBAI);
  Route(:laptop,:BANGKOK,:KUALALUMPUR);
  Route(:laptop,:BANGKOK,:HONGKONG);
```

```
  Route(:laptop,:BANGKOK,:BARCELONA);

  Route(:laptop,:LONDON,:SINGAPORE);
  Route(:laptop,:LONDON,:NEWYORK);
  Route(:laptop,:LONDON,:ISTANBUL);
  Route(:laptop,:LONDON,:DUBAI);
  Route(:laptop,:LONDON,:KUALALUMPUR);
  Route(:laptop,:LONDON,:HONGKONG);
  Route(:laptop,:LONDON,:BARCELONA);

  Route(:laptop,:PARIS,:SINGAPORE);
  Route(:laptop,:PARIS,:NEWYORK);
  Route(:laptop,:PARIS,:ISTANBUL);
  Route(:laptop,:PARIS,:DUBAI);
  Route(:laptop,:PARIS,:KUALALUMPUR);
  Route(:laptop,:PARIS,:HONGKONG);
  Route(:laptop,:PARIS,:BARCELONA);
]

struct Supply
    p::Symbol
    o::Symbol
end
```

8.1 Creating the array `supplies`

```
supplies = Supply[] # Creates a 0 element array of immutable type Supply
for r in routes
    push!(supplies, Supply(r.p, r.o))
end
```

8.2 Creating `suppliedAmount` dictionary

It might be better to create this as a dictionary, where the key is the element of the array `supplies` and the value is the corresponding supplied amount

```
suppliedAmount = Dict{Supply, Float64}()
for s in supplies
    if s.p == :smartphone && s.o == :LONDON
        suppliedAmount[s]=800
        elseif s.p == :smartphone && s.o==:BANGKOK
        suppliedAmount[s]=500
        elseif s.p == :smartphone && s.o==:PARIS
        suppliedAmount[s]=600
        elseif s.p == :tablet && s.o==:BANGKOK
        suppliedAmount[s]=1000
        elseif s.p == :tablet && s.o==:LONDON
        suppliedAmount[s]=1500
        elseif s.p == :tablet && s.o == :PARIS
        suppliedAmount[s]=1700
        elseif s.p == :laptop && s.o == :BANGKOK
        suppliedAmount[s]=150
        elseif s.p == :laptop && s.o == :LONDON
        suppliedAmount[s]=250
        elseif s.p == :laptop && s.o == :PARIS
        suppliedAmount[s]=400
end #if
end #for
```

```
struct Customer
    p::Symbol
    d::Symbol
end
```

8.3 Creating customers array, which is an array of custom immutable `Customer`

```
customers = Customer[]
for r in routes
    push!(customers, Customer(r.p, r.d))
end

demandedAmount = Dict{Customer, Float64}()
for c in customers
#1
    if c.p==:smartphone && c.d==:SINGAPORE
        demandedAmount[c]=400
#2
        elseif c.p==:tablet && c.d==:SINGAPORE
        demandedAmount[c]=600
#3
        elseif c.p==:laptop && c.d==:SINGAPORE
        demandedAmount[c]=90
#4
        elseif c.p==:smartphone && c.d==:NEWYORK
        demandedAmount[c]=200
#5
        elseif c.p==:tablet && c.d==:NEWYORK
        demandedAmount[c]=650
#6
        elseif c.p==:laptop && c.d==:NEWYORK
        demandedAmount[c]=110
#7
        elseif c.p==:smartphone && c.d==:ISTANBUL
        demandedAmount[c]=100
#8
        elseif c.p==:tablet && c.d==:ISTANBUL
        demandedAmount[c]=300
#9
        elseif c.p==:laptop && c.d==:ISTANBUL
        demandedAmount[c]=0
#10
        elseif c.p==:smartphone && c.d==:DUBAI
        demandedAmount[c]=175
#11
    elseif c.p==:tablet && c.d==:DUBAI
        demandedAmount[c]=350
#12
    elseif c.p==:laptop && c.d==:DUBAI
        demandedAmount[c]=65
#13
    elseif c.p==:smartphone && c.d==:KUALALUMPUR
        demandedAmount[c]=550
#14
    elseif c.p==:tablet && c.d==:KUALALUMPUR
```

```
        demandedAmount[c]=950
#15
    elseif c.p==:laptop && c.d==:KUALALUMPUR
        demandedAmount[c]=185
#16
    elseif c.p==:smartphone && c.d==:HONGKONG
        demandedAmount[c]=200
#17
    elseif c.p==:tablet && c.d==:HONGKONG
        demandedAmount[c]=750
#18
    elseif c.p==:laptop && c.d==:HONGKONG
        demandedAmount[c]=150
#19
    elseif c.p==:smartphone && c.d==:BARCELONA
        demandedAmount[c]=275
#20
    elseif c.p==:tablet && c.d==:BARCELONA
        demandedAmount[c]=600
#21
    elseif c.p==:laptop && c.d==:BARCELONA
        demandedAmount[c]=200
    end
end
```

```
costCof =
[34; 7; 8; 10; 11; 74; 9; 18; 5; 15; 6; 23; 81; 18; 20; 10; 9;
 13; 25; 85; 13; 40; 17; 7; 16; 20; 80; 9; 24; 5; 15; 11; 23;
90; 22; 19; 15; 16; 15; 24; 100; 21; 37; 12; 9; 16; 14;
88; 9; 28; 13; 17; 8; 32; 100; 18; 28; 15; 18; 16; 30; 102; 15]
```

```
63-element Array{Int64,1}:
  34
   7
   8
  10
  11
  74
   9
  18
   5
  15
   ⋮
 100
  18
  28
  15
  18
  16
  30
 102
  15
```

8.4 Creating `costRoutes` dictionary which contains the costs of the relevant routes

```
costRoutes=Dict{Route, Float64}()
```

```
for i in 1:length(routes)
    costRoutes[routes[i]]=costCof[i]
end
```

8.5 Creating `orig`, which takes the product as the input and gives the set of origins of that product

```
orig = Dict{Symbol, Array}()
for i in 1:length(products)
    dummy_array = Symbol[]
  for j in 1:length(routes)
        #println(i, j, products[i] == routes[j].p)
        if products[i] == routes[j].p
            push!(dummy_array, routes[j].o)
            #println(orig[products[i]])
        else
        #println("Oops, something is not right")
        end #if
    end #for
    orig[products[i]]=unique(dummy_array)
end #for
```

8.6 Creating `dest`, which takes the product as the input and gives the set of destinations of that product

```
dest = Dict{Symbol, Array}()
for i in 1:length(products)
    dummy_array = Symbol[]
  for j in 1:length(routes)
        #println(i, j, products[i] == routes[j].p)
        if products[i] == routes[j].p
            push!(dummy_array, routes[j].d)
            #println(orig[products[i]])
        else
        #println("Oops, something is not right")
            end #if
        end #for
    dest[products[i]]=unique(dummy_array)
end #for
```

9 JuMP Model

```
using JuMP
using GLPK


transpModel = Model(with_optimizer(GLPK.Optimizer))
@variable(transpModel, trans[routes] >= 0)
@objective(transpModel, Min, sum(costRoutes[l]*trans[l] for l in routes))
```

$$34trans_{Main.WeaveSandBox26.Route(:smartphone,:BANGKOK,:SINGAPORE)}+7trans_{Main.WeaveSandBox26.Route(:smartpho}$$

9.1 First Constraint

```
for pr in products
    for org in orig[pr]
        @constraint(transpModel, sum(trans[Route(pr, org, de)] for de in dest[pr])
                        ==
                        suppliedAmount[Supply(pr,org)])
    end
end
```

9.2 Second Constraint

```
for pr in products
    for de in dest[pr]
        @constraint(transpModel, sum(trans[Route(pr, org, de)] for org in orig[pr])
                        ==
        demandedAmount[Customer(pr,de)])
    end
end
```

9.3 Final constraint:

```
for org in cities
    for de in cities
        if org!=de
        @constraint(transpModel,
        sum(
            trans[r] for r in routes
            if r.o == org && r.d==de # This will be used as an filtering condition
        )
                        <=
        capacity)
        else
            continue
        end
    end
end

statusMipModel = optimize!(transpModel)

println("The optimal objective value is:  ", objective_value(transpModel))

The optimal objective value is: 178330.0

println("The optimal solution is, trans= \n", value.(trans))

The optimal solution is, trans=
1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, Main.WeaveSandBox26.Route[Route(:smartphone, :BANGKOK, :SI
NGAPORE), Route(:smartphone, :BANGKOK, :NEWYORK), Route(:smartphone, :BANGK
OK, :ISTANBUL), Route(:smartphone, :BANGKOK, :DUBAI), Route(:smartphone, :B
ANGKOK, :KUALALUMPUR), Route(:smartphone, :BANGKOK, :HONGKONG), Route(:smar
tphone, :BANGKOK, :BARCELONA), Route(:smartphone, :LONDON, :SINGAPORE), Rou
te(:smartphone, :LONDON, :NEWYORK), Route(:smartphone, :LONDON, :ISTANBUL)
 ...  Route(:laptop, :LONDON, :KUALALUMPUR), Route(:laptop, :LONDON, :HONGKON
G), Route(:laptop, :LONDON, :BARCELONA), Route(:laptop, :PARIS, :SINGAPORE)
, Route(:laptop, :PARIS, :NEWYORK), Route(:laptop, :PARIS, :ISTANBUL), Rout
e(:laptop, :PARIS, :DUBAI), Route(:laptop, :PARIS, :KUALALUMPUR), Route(:la
ptop, :PARIS, :HONGKONG), Route(:laptop, :PARIS, :BARCELONA)]
And data, a 63-element Array{Float64,1}:
```

0.0
0.0
0.0
0.0
500.0
0.0
0.0
355.0
50.0
0.0
175.0
20.0
200.0
0.0
45.0
150.0
100.0
0.0
30.0
0.0
275.0
0.0
0.0
0.0
0.0
0.0
565.0
435.0
0.0
650.0
0.0
350.0
315.0
185.0
0.0
600.0
0.0
300.0
0.0
635.0
0.0
165.0
0.0
0.0
0.0
0.0
150.0
0.0
0.0
35.0
0.0
0.0
65.0
0.0
150.0
0.0
55.0
110.0
0.0

```
  0.0
 35.0
  0.0
200.0
```