Modelos y Optimización I

Trabajo práctico: Problema Combinatorio

de Valais, Ezequiel (94463) Rozanec, Matias (97404)

Octubre 2017

Contents

TRABAJO PRÁCTICO

Problema Combinatorio

IngFraTech está evaluando ahora la ubicación de los datacenters necesarios para proveer los nuevos servicios.

Dado que la mayoría de sus clientes se encuentra en Estados Unidos, deciden ubicar allí los datacenter con la idea de hacer mínima la latencia entre los datacenters y los clientes. Como en un principio no saben de dónde serán sus clientes (es decir, a qué estado pertenecerán) la empresa plantea reducir la latencia global del servicio definiéndolo de la siguiente manera:

"La latencia global del sistema es la suma de las latencias de cada estado. El cálculo de latencia de cada estado se hace considerando la distancia con el datacenter más próximo, estimándose 1 ms de latencia por cada milla de distancia."

Actualmente cuenta con 2 datacenters, uno en Oregon y otro en Florida. Por motivos económicos la empresa no puede agregar más de 3 datacenters nuevos.

Dado que no esperan clientes de Hawái y Alaska, estos estados no son tenidos en cuenta para calcular la latencia global.

¿Qué es lo mejor que puede hacer IngFraTech?

Part I
Descripción de la situación problemática

Se trata de un problema de combinatoria, en el que habrá que incluir variables continuas y booleanas.
En esta instancia podemos afirmar que habrá que considerar una variable *latencia* que deberá ser una variable continua; así como una variable booleana que indique si un determinado datacenter se encuentra instalado en un estado específico.

Part II
Objetivo

Determinar en qué estados van a estar los 3 nuevos datacenters (DB, DC, DD) durante un período de tiempo para minimizar la latencia global del sistema.

Part III
Hipótesis y supuestos

- Se instalarán los 3 datacenters puesto que cada datacenter agregado va a reducir la latencia.
- Se tomará un punto en cada estado. El mismo será referente para calcular las distancias entre estados y las respectivas latencias. No hay opción de instalar un datacenter en otro punto del estado que el mencionado.
- Para el cálculo se consideran únicamente las distancias, no la cantidad de usuarios por estado. O expresado de otra forma: para el modelo, la distribución de usuarios es uniforme a lo largo de todos los estados y en cada uno de los estados.
- No puede haber dos datacenters en un mismo estado.
- Los costos de instalación y mantenimiento de datacenters, así como cualquier otro costo que pueda implicar la instalación de los mismos, no serán tenidos en cuenta por el modelo.

Part IV
Definición de variables, incluyendo unidades

- L_i : variable continua que indica la latencia (en ms) correspondiente al estado i. $\forall i \in [1, 48]$
- DC_i : variable continua que indica la distancia (en millas) del datacenter C al estado i. (ídem para datacenters D y E. DA_i y DB_i son datos conocidos). $\forall i \in [1, 48]$
- YA_i : variable bivalente que vale 0 cuando el estado i tiene la menor latencia. (ídem B, C, D, E)
- YCe_i : variable bivalente que vale 1 cuando el datacenter C está en el Estado i. (ídem D, E)

Constantes

- D_{ij}: distancia entre estado i y estado j.
- M: valor mayor a cualquier distancia posible. Su valor se definirá al momento de pasar el modelo a software.

Part V
Modelo de Programación Lineal Continua

$$\sum_i YAi + YB_i + YC_i + YD_i + YE_i = 1 \tag{1}$$

1 Menor Distancia

Cada L_i tendrá como cota superior la distancia al datacenter más próximo, y como cota inferior esa misma distancia en el caso de que el estado i tenga la menor distancia.
$M \to \infty$

$$DA_i - M * YA_i \leq L_i \leq DA_i, \quad \forall i \in [1, 48] \tag{2}$$

$$DB_i - M * YB_i \leq L_i \leq DB_i, \quad \forall i \in [1, 48] \tag{3}$$

$$DC_i - M * YC_i \leq L_i \leq DC_i, \quad \forall i \in [1, 48] \tag{4}$$

$$DD_i - M * YD_i \leq L_i \leq DD_i, \quad \forall i \in [1, 48] \tag{5}$$

$$DE_i - M * YE_i \leq L_i \leq DE_i, \quad \forall i \in [1, 48] \tag{6}$$

2 Asociación de Datacenter a estado

Se asegura de que cada datacenter pueda ser asignado únicamente a un estado.

$$\sum_{i=1}^{48} YCe_i = 1 \tag{7}$$

$$\sum_{i=1}^{48} YDe_i = 1 \tag{8}$$

$$\sum_{i=1}^{48} YEe_i = 1 \tag{9}$$

3 Asociación de distancia de un estado al datacenter correspondiente

$$DC_i = \sum_{j=1}^{48} D_{ij} * YCe_j \tag{10}$$

$$DD_i = \sum_{j=1}^{48} D_{ij} * YDe_j \tag{11}$$

$$DE_i = \sum_{j=1}^{48} D_{ij} * YEe_j, \quad \forall i \in [1, 48] \tag{12}$$

Part VI
Funcional

$$Z(min) = \sum_{i=1}^{48} L_i \tag{13}$$

Part VII
Heurística

```
#include <iostream>
#include <map>
#include <cfloat>

void get_problem_values(int &num_of_states, int &num_of_datacenters){
    std::cout << "Insert number of states" << std::endl;
    std::cin >> num_of_states;
    std::cout << "Insert number of datacenters" << std::endl;
    std::cin >> num_of_datacenters;
}

void fill_distances_matrix(int num_of_states,
                           std::map<int, std::map<int, double> > &distances){
    for(int i = 1; i <= num_of_states; ++i){
        for(int j = i + 1; j <= num_of_states; ++j){
            double current_distance;
            std::cout << "insert distance between state "
                      << i << " and " << j << std::endl;
            std::cin >> current_distance;
            distances[i][j] = current_distance;
            distances[j][i] = current_distance;
        }
        /* distance from a state to itself is 0 */
        distances[i][i] = 0;
    }
}

double calculate_current_latency(int num_of_states,
                                 const std::map<int, int> &datacenters,
                                 const std::map<int, std::map<int, double> >
                                 &distances){
    double total_latency = 0;
    /* for each state, check the minimum distance to each datacenter */
    for(int i = 1; i <= num_of_states; ++i){
        double this_state_lat = DBL_MAX;
        for(auto it = datacenters.begin(); it != datacenters.end(); ++it){
            double read_lat = distances.at(i).at(it->second);
            if (read_lat < this_state_lat){
                this_state_lat = read_lat;
            }
        }
        total_latency += this_state_lat;
    }
    return total_latency;
}

void manage_manually_allocated_dataceneters(
        int num_of_states, int &num_of_manually_allocated_datacenters,
        std::map<int, int> &datacenters,
        std::map<int, std::map<int, double> > &distances){
    std::cout << "How many datacenters are manually allocated?" << std::endl;
    std::cin >> num_of_manually_allocated_datacenters;
```

```
53
54      for (int i = 1; i <= num_of_manually_allocated_datacenters; ++i){
55          int state;
56          std::cout << "Insert number of state where datacenter " << i
57                    << " should be." << std::endl;
58          std::cin >> state;
59          datacenters[i] = state;
60          /* display current latency */
61          std::cout << "latency after datacenter " << i
62                    << " was allocated: "
63                    << calculate_current_latency(num_of_states, datacenters,
64                                                 distances) << std::endl;
65      }
66  }
67
68  bool state_already_has_datacenter(int state,
69                                    const std::map<int, int> &datacenters){
70      for (auto it = datacenters.begin(); it != datacenters.end(); ++it){
71          if (it->second == state){
72              return true;
73          }
74      }
75      return false;
76  }
77
78  int locate_new_datacenter(int num_of_datacenter,
79                            int num_of_states, std::map<int, int> &datacenters,
80                            const std::map<int, std::map<int, double> >
81                            &distances){
82      /* returns number of state which minimizes latency */
83      double current_lat = DBL_MAX;
84      int state_that_minimizes_lat;
85      for(int i = 1; i <= num_of_states; ++i){
86          if (!state_already_has_datacenter(i, datacenters)){
87              datacenters[num_of_datacenter] = i;
88              if (calculate_current_latency(num_of_states, datacenters,
                    distances)
89                  < current_lat){
90                  state_that_minimizes_lat = i;
91              }
92          }
93      }
94      return state_that_minimizes_lat;
95  }
96
97  void locate_remaining_datacenters(int num_of_states, int num_of_datacenters,
98                                    int num_of_manually_allocated_datacenters,
99                                    std::map<int, int> &datacenters,
100                                   std::map<int, std::map<int, double> >
101                                   &distances){
102     for (int i = num_of_manually_allocated_datacenters + 1;
103          i <= num_of_datacenters; ++i){
104         datacenters[i] = locate_new_datacenter(i, num_of_states,
105                                                datacenters, distances);
106         /* display current latency */
107         std::cout << "latency after datacenter " << i << " located: " <<
108                   calculate_current_latency(
```

```
109                             num_of_states, datacenters, distances) << std::endl;
110     }
111 }
112
113 void inform_datacenter_positions(const std::map<int, int> &datacenters){
114     for (auto it = datacenters.begin(); it != datacenters.end(); ++it){
115         std::cout << "datacenter\t" << it->first <<
116                   "\tlocated_in_state\t" << it->second << std::endl;
117     }
118 }
119
120 int main(){
121     int num_of_states;
122     int num_of_datacenters;
123
124     /* A map is used to store de distance matrix.
125      * States are identified with numbers from 1 to n,
126      * where n is the number of states to consider. */
127     std::map<int, std::map<int, double> > distances;
128
129     /* A map is used to link each datacenter to a state.
130      * K = datacenter.
131      * V = state.  */
132     std::map<int, int> datacenters;
133
134     get_problem_values(num_of_states, num_of_datacenters);
135
136     fill_distances_matrix(num_of_states, distances);
137
138     int num_of_manually_allocated_datacenters;
139     manage_manually_allocated_dataceneters(
140             num_of_states, num_of_manually_allocated_datacenters, datacenters,
141             distances);
142
143     locate_remaining_datacenters(num_of_states, num_of_datacenters,
144                                  num_of_manually_allocated_datacenters,
145                                  datacenters, distances);
146
147     /* inform results */
148     std::cout << "*** Final datacenter positions ****" << std::endl;
149     inform_datacenter_positions(datacenters);
150     return 0;
151 }
```