Proyecto: Cobertura de vértices mínima con colonia de hormigas

Heurísticas de Optimización Combinatoria

Alumna:

Karla Adriana Esquivel Guzmán

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

16/Diciembre/2018

Como segundo proyecto del curso elegí hacer una implementación para encontrar **la cobertura de vértices mínima con una colonia de hormigas** el lenguaje de programación que se utilizó para este proyecto es **Go** y para generar las gráficas se utilizó **SVG**.

- **Problema:** Encontrar una cubierta en una gráfica G con el menor número de vértices posible.
- **Go:** Nuevamente utilicé el lenguaje de programación Go para adquirir más experiencia con este lenguaje.
- **SVG:** SVG es un formato gráfico basado en XML, decidí gráficar con SVG porque es muy fácil de utilizar y dibuja de manera bonita las gráficas.
- **Implementación:** Es una implementación sencilla, se hizo con base en el paper **An Ant Colony Optimization Algorithm for the Minimum Weight Vertex Cover Problem**, la dificultad principal fue entender como funcionaban los dos tipos de feromonas, puesto que hay una global y una local, una vez entendido lo anterior la implementación del algoritmo no tuvo gran dificultad.
- **Estructuras**:
  Esta estructura reprenta un vértice, cada vértice debe tener un valor de feromona inicial y un valor de la feromona que se modificará con cada iteración.

```
type Vertex struct {
  index int
  pheromone_init float64
  pheromone float64
}
```

  Esta estructura representa una arista, cada arista tiene dos valores los cuales son los índices de los vértices en los que incide, así como también debe tener un peso inicial.

```
type Edge struct {
  v1_index int
  v2_index int
  weight float64
}
```

Esta estructura representa una gráfica, contiene un **slice** de veritces, que es un aputnador para que todas las gráficas que se definan para las hormigas hagan referencia al mismo **slice**.

```
type Graph struct {
  vertexes *[]Vertex
  edges *[]Edge
  full *[]Edge
}
```

Esta estructura representa una hormiga, la cual tiene como parámetros un índice index, el cuál sirve como un identificador para la hormiga, tenemos también el apuntador a una gráfica graph, un apuntador a un generador aleatorio random(genera números aleatoriamente), solution es un **slice** de enteros para guardar las soluciones de cada hormiga. Un parámetro q0 que es un valor fijo definido para experimentación, tenemos evaporation_rate esté parametro se utiliza para ajustar la actualización global y pheromone_adjust para ajusta la feromona local, por último tenemos el parámetro beta que es un valor fijo definido desde el inicio para experimentación, sirve para "rankear" (darle cierto peso) a la feromona.

```
type Ant struct {
  index int
  graph *Graph
  random *rand.Rand
  solution *[]int
  q0 float64
  evaporation_rate float64
  pheromone_adjust float64
  beta float64
 }
```

- **Funciones:**
  Utilizando la fórmula especificada en el paper, esta función hace la actualización de la feromona local después de que la hormiga se posiciona sobre el vértice.

```
func (a Ant) ActualizaFeromonaLocalmente(indexVertex int){
pheromone0 := (*(*a.graph).vertexes)[indexVertex].pheromone_init
pheromonei := (*(*a.graph).vertexes)[indexVertex].pheromone
(*(*a.graph).vertexes)[indexVertex].pheromone =
((1-a.pheromone_adjust)*pheromonei)+(a.pheromone_adjust*pheromone0
```

  Esta función simula un "paso" de la hormiga, es decir a que nodo será el siguiente al que avanzará, viene explicitamente comentada cada una de las lineas en el código.

```
func (a Ant) Paso() {
    q := a.random.Float64()
    rnvalues := a.GetVertexesRNValue()
    sumrnValues := 0.0
    maxrnValue := 0.0
    indexmaxrnValue := -1
    count := 0;
    for count < len(rnvalues) {
        if maxrnValue <= rnvalues[count] {
            maxrnValue = rnvalues[count]
            indexmaxrnValue = count
        }
        sumrnValues = sumrnValues + rnvalues[count]
        count = count+1
    }
    if q < a.q0{
    [indexmaxrnValue].pheromone)
    a.AgregaASolucion(indexmaxrnValue)
    a.ActualizaFeromonaLocalmente(indexmaxrnValue)
    (*a.graph).FullEdgesToOForVertex(indexmaxrnValue)
    [indexmaxrnValue].pheromone)


    }else{
        dioPaso := false
        for dioPaso != true{
            randomIndex := rand.Intn(len(rnvalues))
            vertice
```

```
            indexProb := (rnvalues[randomIndex]/sumrnValues)
            randonNumber := a.random.Float64()
            indexProb, randonNumber)
            if randonNumber <= indexProb{
                a.AgregaASolucion(randomIndex)
                a.ActualizaFeromonaLocalmente(randomIndex)
                (*a.graph).FullEdgesToOForVertex(randomIndex)
                dioPaso = true
            }
        }
    }
}
```

Esta función sirve para determinar si la hormiga puede continuar avanzando en las aristas, si estás tienen un valor distinto de 0.

```
func (a Ant) PuedeDarUnPaso() bool {
    peso := (*a.graph).FullWeight()
    if peso != 0{
        return true
    }else{
        return false
    }
}
```

Esta función es la encargada de calcular el valor RN de cada uno de los vértices.

```
  func (a Ant) GetVertexesRNValue() []float64 {
    numVertex := len((*(*a.graph).vertexes))
    rnvalues := make([]float64, numVertex)
    count := 0;
    for count < numVertex {
        indexVertex := (*(*a.graph).vertexes)[count].index
        pheromoneVertex := (*(*a.graph).vertexes)[count].pheromone
        weightVertex := (*a.graph).FullWeightOfVertex(indexVertex)
        rnvalues[indexVertex] = pheromoneVertex*weightVertex
        indexVertex,pheromoneVertex,weightVertex, rnvalues[indexVertex])
        count = count+1
    }
    return rnvalues
 }
```

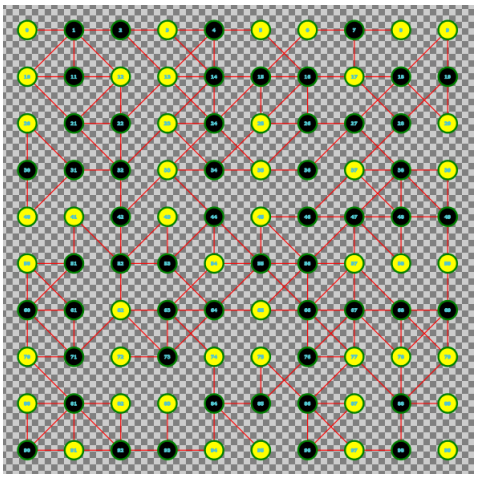
Ahora explicaré en esta parte la función main que es la más importante de todas, porque se inicializa la gráfica y es en dónde está definida la estructura del algoritmo ACO tal y como está definido en el paper ya mencionado, en el código viene comentado lo que hace cada parte del código.

```
func main() {
  data, err := ioutil.ReadFile("g.txt")
  if err != nil {
      fmt.Println("File reading error", err)
      return
  }
  rows := strings.Split(string(data), "\n")
  numOfVertexes, _ := strconv.Atoi(rows[0])
  vertexesG := make([]Vertex, numOfVertexes)
  count := 0;
  for count < numOfVertexes {
      vertexesG[count] = Vertex{count, 0.1, 0.2}
      count = count + 1
  }

  numOfEdges := len(rows)-2
  edgesG := make([]Edge, numOfEdges)
  count = 0;
  for count < numOfEdges {
      vertIndx := strings.Split(rows[count+1], ",")
      vertex1, _ := strconv.Atoi(vertIndx[0])
      vertex2, _ := strconv.Atoi(vertIndx[1])
      edgesG[count] = Edge{vertex1,vertex2,1}
      count = count + 1
  }
```

Experimentación y Resultados

Cada semilla genera una gráfica aleatoria, y esta misma semilla se utiliza para generar los números aleatorios que se utilizan en el algoritmo ACO. Al terminar cada ciclo, se guarda la mejor solución que se encuentra, en cada ejecución para cada gráfica dentro de la carpeta GraficaSeed cada iteración se realizó con **20 hormigas** y con el valor de la pheromone_init(feromona inicial) de 0.1. La siguiente gráfica generada por la semilla 0 con su cobertura mínima, en dónde la cobertura mínima se marca de color negro.



Para consultar las demás semillas y replicar el experimento, basta con buscar en la carpeta GraficaSeed en dónde se muestran en formato .txt las gráficas y a su lado se encuentra el SVG de cada una de las soluciones.
Hay 2 archivos llamados **main.go** y **mainrandomlatiz.go** la primera corre el ejemplo del PDF que envió Luis, y la segunda versión genera las gráficas aleatoriamente, hay una parte del código que puede comentarse y "descomentarse" para quitar o dejar ciclos, cuando dejas los ciclos se genera par 50 gráficas diferentes y el algoritmo se ejecuta sobre las 50 diferentes gráficas y como semillas se usan los números enteros entre 0-49 ahorita como lo deje en la última versión es para que se genere una sola gráfica.

Bibliography

Shyong Jian Shyu, An Ant Colony Optimization Algorithm for the Minimum Weight Vertex Cover Problem, Department of Computer Science and Information Engineering, Ming Chuan University, 2004.