Práctica 3: Algoritmos Greedy

Alejandro Rodríguez López
alexrodriguezlop@correo.ugr.es

29 de abril de 2018

Índice general

Capítulo 0

Introducción

0.1. Presentación del problema

Un electricista necesita hacer n reparaciones urgentes, y sabe de antemano el tiempo que le va a llevar cada una de ellas: en la tarea i ésima tardará t_i minutos. Como en su empresa le pagan dependiendo de la satisfacción del cliente y esta es inversamente proporcional al tiempo que tardan en atenderles, necesita decidir el orden en el que atenderá los avisos para minimizar el tiempo medio de atención de los clientes (desde el inicio hasta que su reparación es efectuada).

- Diseñar un algoritmo Greedy para resolver esta tarea. Demostrar que el algoritmo obtiene la solución óptima.
- Modificar el algoritmo anterior para el caso de una empresa en la que se disponga de los servicios de más de un electricista.

Capítulo 1

Algoritmos voraces o Algoritmos Greedy

Es una estrategia de búsqueda por la cual se sigue una heurística consistente en elegir la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima.
Este esquema algorítmico es el que menos dificultades plantea a la hora de diseñar y comprobar su funcionamiento.
Normalmente se aplica a los problemas de optimización.

1.1. Esquema Greedy

Dado un conjunto finito de entradas C, un algoritmo voraz devuelve un conjunto S (seleccionados) tal que $S \sqsubseteq C$ y que además cumple con las restricciones del problema inicial.
A cada conjunto S que satisfaga las restricciones se le suele denominar prometedor, y si este además logra que la función objetivo se minimice o maximice (según corresponda) diremos que S es una solución óptima.

Elementos de los que consta la técnica:

- El conjunto C de candidatos, entradas del problema.
- Función de selección. Informa cuál es el elemento más prometedor para completar la solución. Éste no puede haber sido escogido con anterioridad. Cada elemento es considerado una sola vez. Luego, puede ser rechazado o aceptado y pertenecerá a $C \setminus S$.
- Función de factibilidad. Informa si a partir de un conjunto se puede llegar a una solución. Lo aplicaremos al conjunto de seleccionados unido con el elemento más prometedor.
- Función solución. Comprueba, en cada paso, si el subconjunto actual de candidatos elegidos forma una solución (no importa si es óptima o no lo es).
- Función objetivo. Es aquella que queremos maximizar o minimizar, el núcleo del problema.

Funcionamiento:

El algoritmo escoge en cada paso al mejor elemento $x \in C$ posible, conocido como el elemento más prometedor. Se elimina ese elemento del conjunto de candidatos ($C \leftarrow C \setminus \{x\}$) y, acto seguido, comprueba si la inclusión de este elemento en el conjunto de elementos seleccionados ($S \cup \{x\}$) produce una solución factible.

En caso de que así sea, se incluye ese elemento en S. Si la inclusión no fuera factible, se descarta el elemento. Iteramos el bucle, comprobando si el conjunto de seleccionados es una solución y, si no es así, pasando al siguiente elemento del conjunto de candidatos.

Capítulo 2

Solución al problema

2.1. Solución Greedy un único electricista

2.1.1. Planteamiento de la solución

Planteo la solución mas óptima asignando las tareas de forma creciente en función a su tiempo de ejecución.

Esquema Greedy:

- Consideramos el conjunto de candidatos como las tareas a ejecutar.
- Función Selección: Elegiremos siempre la tarea cuya realización conlleve un menor tiempo.
- Función de factibilidad: Valoramos que la tarea no haya sido seleccionada previamente.
- Función de solución: Deben haber sido asignadas todas las tareas.
- Función objetivo: Asignar todas las tareas de la forma mas óptima posible entendiendo ésta como la ordenación que produce el tiempo de espera mínimo

2.1.2. Demostración

Cuadro 2.1: Tareas y tiempos

	T1	**T2**	**T3**
E1	5	10	3

Consideramos las n tareas de la tabla 2.1 con sus respectivos tiempos.
Si valoramos todas las permutaciones posibles obtenemos los resultamos mostrados en la tabla 2.2.

Cuadro 2.2: Permutaciones posibles

T1 (5)	**T2 (10)**	**T3 (3)**	$\sum_t^{t_n}$
T1	T2	T3	5+(5+10)=20
T1	T3	T2	5+(5+3)=13
T2	T1	T3	10+(10+5)=25
T2	T3	T1	10+(10+3)=23
T3	T1	T2	3+(3+5)=11
T3	T2	T1	3+(3+10)=16

Como se puede observar atendiendo a la tabla 2.2 la solución del problema en este caso, seria la secuencia de tareas (T3,T1,T2), que corresponde con la ordenación de modo creciente de los tiempos de las tareas.
También podemos observar que ante cualquier permutación de estas tareas que desordene la solución optima seleccionada, devuelve un tiempo de espera mayor que esta.
El algoritmo Greedy propuesto nos da una solución que coincide con la óptima por lo que el criterio de selección propuesto siempre conduce a una solución óptima.

2.1.3. Solución alternativa

El problema planteado también podría resolverse ordenando el vector de tareas en orden ascendente en relación con el tiempo de ejecución de cada tarea y posteriormente asignarlas en dicho orden.
Aunque cabe destacar que ésta solución no estaría basada en algoritmos voraces o greedy.

2.1.4. Código

```
/**
@brief Obtiene el orden de ejecucion de tareas mas optimo en funcion al tiempo de ejecucion de las mismas.
@param v: vector de pares que contiene el numero de tarea y su tiempo correspondiente.
*/
void optimizacionTareas(vector<pair<int, int>> & conjuntoTareas){
    vector<pair<int, int>> solucion;
    vector<pair<int, int>>::iterator tarea;


    //(Mientras queden candidatos y no se haya encontrado la solucion)
    while(!conjuntoTareas.empty()){

        //Seleciono una tarea
        tarea = tareaMenortiempo(conjuntoTareas);

        //Sí la tarea es factible, la incluyo en la solucion
        solucion.push_back(*tarea);


        //La elimino del vector conjuntoTareas
        conjuntoTareas.erase(tarea);
    }

    conjuntoTareas = solucion;
}
```

Figura 2.1: Código del algoritmo para un electricista

2.1.5. Resultados

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
ar0@ar0-ubuntu:~/Escritorio/ALG/P3$ g++ -o practica P3.cpp -std=gnu++0x
ar0@ar0-ubuntu:~/Escritorio/ALG/P3$ ./practica 10
##########**Conjunto de candidatos**############
Tarea 1         Tiempo de ejecucion:6
Tarea 2         Tiempo de ejecucion:12
Tarea 3         Tiempo de ejecucion:14
Tarea 4         Tiempo de ejecucion:10
Tarea 5         Tiempo de ejecucion:6
Tarea 6         Tiempo de ejecucion:10
Tarea 7         Tiempo de ejecucion:12
Tarea 8         Tiempo de ejecucion:4
Tarea 9         Tiempo de ejecucion:18
Tarea 10                Tiempo de ejecucion:2

##########**Resultado**############
Tarea 10                Tiempo de ejecucion:2
Tarea 8         Tiempo de ejecucion:4
Tarea 1         Tiempo de ejecucion:6
Tarea 5         Tiempo de ejecucion:6
Tarea 4         Tiempo de ejecucion:10
Tarea 6         Tiempo de ejecucion:10
Tarea 2         Tiempo de ejecucion:12
Tarea 7         Tiempo de ejecucion:12
Tarea 3         Tiempo de ejecucion:14
Tarea 9         Tiempo de ejecucion:18

ar0@ar0-ubuntu:~/Escritorio/ALG/P3$
```

Figura 2.2: Resultados de la ejecución del código para un solo electricista

2.1.6. Eficiencia

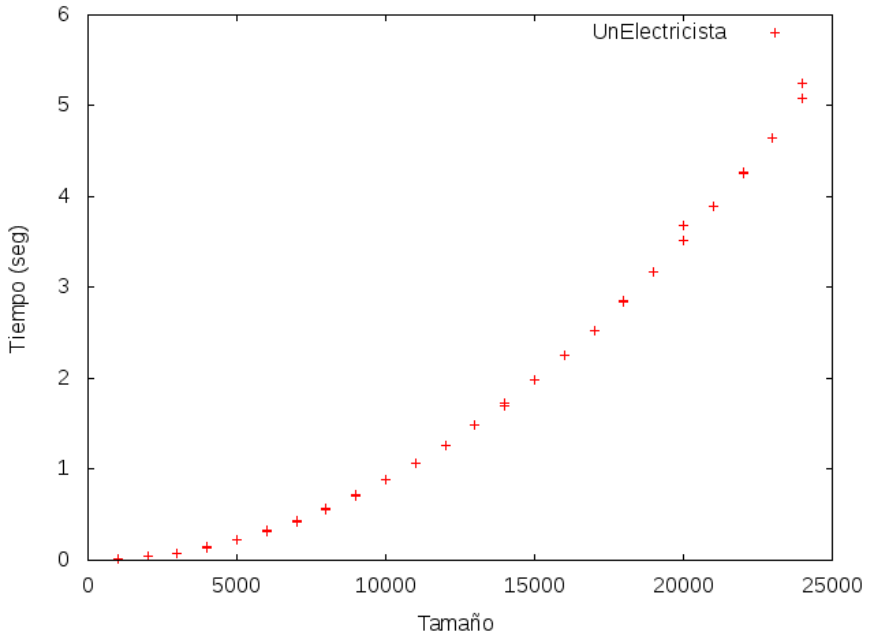

Figura 2.3:

2.2. Solución Greedy varios electricistas

2.2.1. Planteamiento de la solución

Planteo la solución asignando la tarea en orden de tiempo de ejecución de mayor a menos a aquel electricista cuya última tarea asignada tenga el tiempo de ejecución mas pequeño y ademas tenga el menor número de tareas asignadas.
De esta forma se obtendrá la solución mas óptima al problema planteado.
Esquema Greedy:

- Consideramos el conjunto de candidatos como las tareas a ejecutar, aunque también se podrían seleccionar los electricistas.
- Función Selección: Elegiremos siempre la tarea cuya realización conlleve un mayor tiempo para poder hacer el reparto de forma equitativa.
- Función de factibilidad: Valoramos que la tarea no haya sido seleccionada previamente.
- Función de solución: Deben haber sido asignadas todas las tareas de forma equitativa entre los electricistas.
- Función objetivo: Asignar todas las tareas de la forma mas óptima posible entendiendo ésta como la ordenación que produce el tiempo de espera mínimo

2.2.2. Demostración

Asignaremos la tarea en orden de tiempo de ejecución de mayor a menos a aquel electricista cuya última tarea asignada tenga el tiempo de ejecución mas pequeño y ademas tenga el menor número de tareas asignadas.

De esta forma es reparto será equitativo y se encontrará la solución mas óptima.

Cuadro 2.3: Reparto de tareas 1

	T1 (5)	T2 (10)	T3 (3)	T4 (20)	T5 (14)	T6 (18)	T7 (32)	T8 (10)	Sum
E1			x				x		35
E2	x			x					25
E3		x				x			28
E4					x			x	24

En la tabla 2.3 podemos observar como repartiendo las tareas con el método anteriormente descrito el reparto se hace de manera equitativa.

Cuadro 2.4: Reparto de tareas 2

	T1 (5)	T2 (10)	T3 (3)	T4 (20)	T5 (14)	T6 (18)	T7 (32)	T8 (10)	Sum
E1			x		x				17
E2	x					x			23
E3		x		x					30
E4							x	x	42

En la tabla 2.4 podemos observar como repartiendo las tareas de forma ascendente y rotativa el reparto no llega a ser equitativo.

2.2.3. Código

```
/**
@brief Obtiene el orden de ejecucion de tareas mas optimo en funcion al tiempo de ejecucion de las mismas.
@param v: vector de pares que contiene el numero de tarea y su tiempo correspondiente.
@param numTareas: numero de tareas.
*/
vector<stack<pair<int, int>>> optimizacionTareas(vector<pair<int, int>> & conjuntoTareas, int numElectricistas){
    vector<stack<pair<int, int>>> solucion;
    vector<pair<int, int>>::iterator tarea;
    vector<stack<pair<int, int>>>::iterator lectura, insercion, ultimaInsercion;

    //Inicializacion de solucion
    stack<pair<int, int>> aux;
    aux.push(make_pair(0,0));

    for (int i=0; i<numElectricistas; i++)
    {
        solucion.push_back(aux);
    }
    ultimaInsercion = solucion.begin();
```

Figura 2.4: Código del algoritmo para varios electricistas

```
    //(Mientras queden candidatos y no se haya encontrado la solucion)
    while(!conjuntoTareas.empty()){

        insercion = solucion.begin();
        lectura = ++ solucion.begin();

        //Seleciono una tarea
        tarea = tareaMayorTiempo(conjuntoTareas);

        //Buscamos a que electricista asignar la tarea.
        while(lectura != solucion.end()){
            //Si la tarea es factible, la incluyo en la solucion
            //Comprueba entre aquellos en los que no se a llevado a cabo la ultima insercion
            if((*lectura).size() < (*ultimaInsercion).size() && (*lectura).top().second < (*insercion).top().second){
                insercion = lectura;
            }
            ++ lectura;
        }
        //Asignamos la tarea a la cola del electricista seleccionado.
        (*insercion).push(*tarea);

        //Guardamos el ultimo que recibio tarea
        ultimaInsercion = insercion;

        //La elimino del vector conjuntoTareas
        conjuntoTareas.erase(tarea);
    }
    return solucion;
}
```

Figura 2.5: Código del algoritmo para varios electricistas

2.2.4. Resultados

```
ar0@ar0-ubuntu:~/Escritorio/ALG/P3$ ./practica 5 2
##########**Conjunto de candidatos**############
Tarea 1         Tiempo de ejecucion:7
Tarea 2         Tiempo de ejecucion:9
Tarea 3         Tiempo de ejecucion:3
Tarea 4         Tiempo de ejecucion:5
Tarea 5         Tiempo de ejecucion:1

##########**Resultado**############
------Electricista 1--------
Tarea 5         Tiempo de ejecucion:1
Tarea 3         Tiempo de ejecucion:3
Tarea 4         Tiempo de ejecucion:5
Total tiempo en tareas: 9

------Electricista 2--------
Tarea 1         Tiempo de ejecucion:7
Total tiempo en tareas: 7

ar0@ar0-ubuntu:~/Escritorio/ALG/P3$
```

Figura 2.6: Resultados de la ejecución del código para varios electricistas

2.2.5. Eficiencia

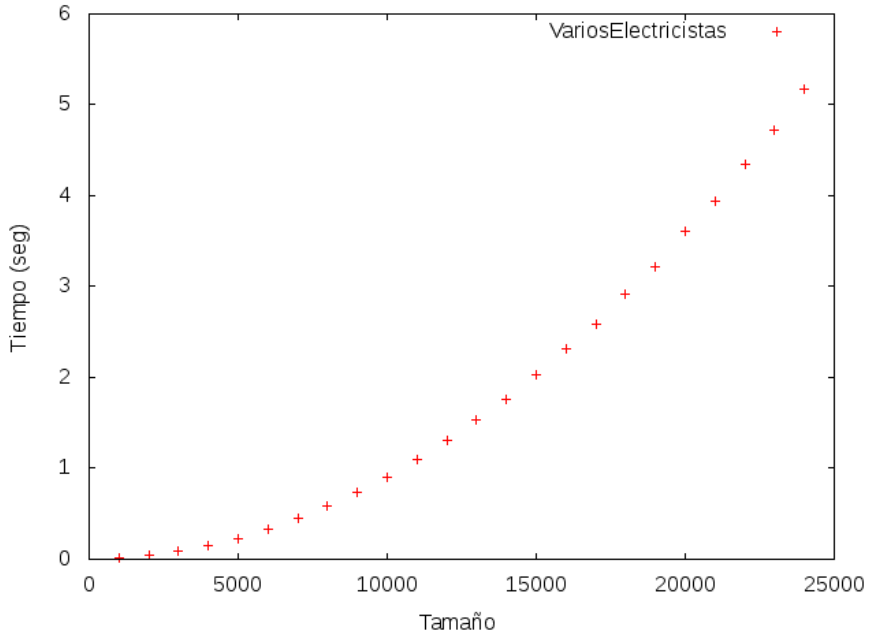

Figura 2.7:

Capítulo 3

Conclusiones

Como conclusión darnos cuenta de que Greedy no garantiza una solución óptima, pues un mismo problema puede ser resuelto de varias formas o simplemente no ser aplicable a ésta metodología.

Bibliografía

[1] Wikipedia, "Algoritmo voraz o Greedy", https://es.wikipedia.org/wiki/Algoritmo_voraz