

# Aplicación de algoritmo Goloso para problema de optimización de costos

Maximiliano Araya Poblete (5.5 horas) y Miguel Salinas González (5 horas)

*Departamento de Ingeniería Informática*

*Universidad de Santiago de Chile, Santiago, Chile*

maximiliano.araya@usach.cl, miguel.salinas@usach.cl

**Resumen**—Este artículo consta de un breve paseo por el conocido algoritmo llamado Goloso (o Greedy) este es aplicado a un problema de optimización, básicamente este problema plantea el cómo transportar múltiples cargas con el menor costo asociado. Se obtiene de la experiencia un algoritmo polinomial programado en C con el uso de herramientas de makefile y debug.

**Palabras claves**—Algoritmo, Goloso, Orden de complejidad, Makefile y Optimización.

obstante de no hallarse el óptimo retornara una solución mas que aceptable, una buena solución cercana al óptimo. A pesar de no asegurar un óptimo es utilizado pues su esquema algorítmico es simple y no plantea muchas dificultades en comparación a por ejemplo el Backtracking, que a pesar de asegurar el óptimo, su implementación es mas complicada en comparación.

## I. INTRODUCCIÓN

Las ciencias de la computación se enfrentan desde sus inicios a problemas de optimización, con el paso de los años múltiples métodos se diseñaron para la resolución de problemas. Uno de los métodos mas conocidos es el Goloso pues encuentra una buena solución (no asegura el óptimo) a través de la generación de una combinación de soluciones óptimas para cada iteración o "paso local" con la esperanza de que esta combina sea o se acerque al óptimo. Por ello y para terminar su aprendizaje se le encomienda a los alumnos de ingeniería informática de la universidad de santiago de chile la creación de un software para el cliente Clover Sanity, esta empresa posee múltiples centros de acopio o "basurales" donde desechan la basura de sus servicios de limpieza, por otro lado el gobierno dispone de incineradores donde quemar esta basura pero le solicita la empresa que tengan toda su basura en una cantidad limitada de basurales, específicamente tantos basurales como incineradores y la empresa se enfrenta al problema de trasladar la basura de los basurales, el problema es que buscan reducir el costo asociado a esta acción y logrando dejar la basura en la cantidad de basurales indicada, el software a diseñar debe solucionar el mencionado problema con el uso del Goloso y la herramienta del Makefile. Se busca que el estudiante cierre el ciclo de aprendizaje del mencionado método con la aplicación practica anteriormente descrita.

## II. DESCRIPCIÓN DEL MÉTODO

### II-A. Goloso

El Goloso o Greedy es una estrategia de búsqueda en la cual se escoge la opción óptima en cada iteración o paso, con la esperanza de que esa combinación de opciones entregue finalmente la solución óptima. Lamentablemente esta estrategia no asegura encontrar la solución óptima, no

## III. DESCRIPCIÓN DE LA SOLUCIÓN

### III-A. Lista de pasos

1. Análisis del problema: comprensión del problema, definición de funcionalidades y generación de solución a través de método goloso.
2. Conclusión del análisis del problema: se concluye que la solución sera generar todas las combinaciones posibles, donde cada combinación es el movimiento de basura de un basural a otro, se escoge la de menor costo, se lleva a cabo el movimiento, y luego repitiendo hasta reducir el numero de basurales con basura a el numero de incineradores. Además al analizar la matemática función de costo se descubre que el subsidio dado por el gobierno no es relevante pues todos lo tienen por lo que no es necesario considerarlo para el costo de los movimientos mas que al momento de entregar la solución final, específicamente los costos se calculan entre el valor absoluto de la distancia inicial menos la final, todo multiplicado por las toneladas de basura (dato que fue factorizado del resto de la formula), dejando que a parte de la formula asociada al subsidio sea otro factor en la multiplicación anterior, el cual seria 1 menos el resultado de 1 dividido el subsidio.
3. Lectura de archivos de entrada: función que recibe el nombre del archivo (archivo de entrada descrito por enunciado) y guarda el contenido en una estructura de datos "basural".
4. Definir máximo de iteraciones: para saber cuantas veces se debe repetir lo descrito en el paso dos se debe restar el numero de basurales con el numero de incineradores, así se obtiene cuantos basurales se deben dejar sin basura.
5. Definir estructura para el Goloso: para trabajar los datos dentro del algoritmos se usan dos estructuras de datos, la mencionada en el paso numero tres, y una estructura llamada "movimiento" la cual se usa para almacenar la

información de las combinaciones, es decir, el basural desde el cual se traslada la basura, aquel al que va la basura y el costo asociado.

6. Funciones asociadas al Goloso: la función desarrollada se apoya y esta conformada de tres otras funciones. "baratisimo" encuentra el movimiento de menor costo de una lista de movimientos. "menores" genera todas las combinaciones para una lista de basurales y retorna una lista de movimientos donde cada uno de ellos es el de menor costo para cada basural considerado como inicial (aquellos de los que se puede mover basura). "tomarMovimiento" lleva a cabo un movimiento, es decir, aquel movimiento de basural de un basural inicial a un basural final que se indica dentro de la estructura movimiento es llevado a cabo dentro de los basurales, es decir, inicial queda sin basural y final contiene la basura de ambos.
7. Desarrollar el Goloso: se crea el Goloso, utiliza la estructura definidas, esto es, una arreglo de basurales, uno de movimientos que contiene todos los movimientos llevados a cabo ("salida") y un movimiento auxiliar, dentro de un ciclo que tiene tantas iteraciones como las establecidas por lo descrito en el paso cuatro, se obtiene la lista de menores para los basurales actuales, se guarda en el movimiento auxiliar aquel movimiento de menor costo de los menores obtenidos, se lleva a cabo este movimiento y es guardado en la lista de movimientos de salida. Básicamente se están generando todas las combinaciones y escogiendo la de menor costo para cada iteración, actualizando los datos de los basurales y evitando en caer ciclos como traspasar la basura de un basural A a un B y luego en la siguiente iteración de B a A, luego esto se repita constantemente.
8. Escritura de la solución: se crea una función que escriba un archivo con la solución, su implementación es trivial pero es necesario indicar que el costo total se calcula sumando los costos de los movimientos realizados del arreglo "salida" luego son multiplicados por el resto de la fórmula de costo, aquella asociada al subsidio.
9. Pruebas del algoritmo, corrección de errores: uno de los principales errores es el ciclo infinito que puede darse al moverse de una basural A a B y luego de B a A en un ciclo que no acaba, lo cual lleva a una solución errada pues generalmente no cumple con el requisito del número de basurales con basura final, lo cual es corregido con condicionales que no permiten que esto se lleve a cabo permitiendo que el resultado entregado sea correcto.
10. Printcurrent: se crea la función print current para el modo Debug del makefile.
11. Documentación.

### III-B. Orden de complejidad

Para el cálculo del orden de complejidad de un algoritmo, es importante saber que en este proceso, se tiene en cuenta siempre, el peor de los casos posibles a resolver por este

algoritmo. Esto último es importante tenerlo en cuenta, ya que es la metodología que se utiliza para calcular la complejidad del algoritmo en cuestión. El algoritmo principalmente consiste de una función llamada goloso, la cual es iterativa, y tiene un llamado a la función menores, la cual también es iterativa y de orden  $n$  al cuadrado. Por lo que para el cálculo de complejidad del algoritmo se considera que menores es ejecutado de forma iterativa, dando como resultado que el orden del algoritmo es  $n$  al cubo, al ser la única función dentro del main, se considera como el orden de complejidad de todo el programa.

## IV. ANÁLISIS DE LA SOLUCIÓN

### IV-A. Tiempos de ejecución

El orden de complejidad nos indica el tiempo para el peor caso de uso del algoritmo, pero, ¿Cuál es el peor caso?, en un problema complejo es difícil saber que caso es peor, no solo depende del tamaño de la entrada, sino, de la cantidad de basura albergada inicialmente en cada basural, pues ciertos basurales pueden encontrarse sin basura de forma que se cambia un poco el contexto, por otro lado se pueden tener casos con una gran entrada de datos, pero que sólo baste un movimiento para entregar la solución. Existen varias aristas al momento de analizar el tiempo de ejecución, puesto que no varía solo del tamaño de entrada aunque claramente es una gran influencia, por ello se realiza la siguiente traza (ver tabla I) de ejecuciones con entradas de  $n$  basurales y 50 toneladas de basura en cada uno.

Tabla I: Traza de ejecución

Tamaño de $n$	tiempo de ejecución en ms
50	1
75	2
100	5
200	40
300	136

Los resultados en cuestión fueron obtenidos en un computador con un procesador Intel i5-4670 junto con 16gb de Ram[?] DDR3 a 1333MHz.

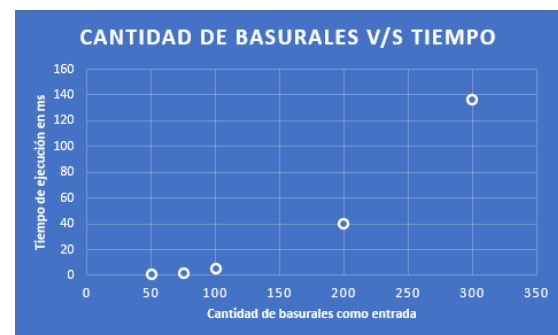


Figura 1: Gráfica de la tabla

Así es como se modela la tabla en un gráfico y a simple vista se puede ver su similitud con una gráfica de una función

cúbica en el primer cuadrante del plano cartesiano, si bien se pudiese apegar a una gráfica de ese estilo, esto no siempre se cumple, ya que se tienen diversos casos, si bien el orden indicado al que se llegó es  $n$  al cubo, hay que tener presente la consideración que se propuso al comienzo de este cálculo, la cuál consiste es que esto es respecto al peor de los casos, y no todos los casos corresponde al peor. Con esto se concluye que sí afecta el tamaño de la entrada en el tiempo de la ejecución, si se tuviese un tamaño muy grande, como puede ser un cantidad de 100.000 basurales, se iría a un tiempo real de demora (es decir, que los milisegundos se transformen en segundos).

#### IV-B. Análisis implementación

El código diseñado es una secuencia de pasos que al finalizar halla la solución a un problema por lo que puede ser clasificado como un algoritmo. El algoritmo desarrollado posee un bucle secundario llamado menores el cual genera los posibles movimientos de basura junto con su costo asociado, este bucle se utiliza dentro de otro bloque llamado goloso, el cual básicamente toma el movimiento de menor costo y lo ejecuta. Así, este algoritmo en cuestión vendría siendo el resultado de una división en subproblemas que tiene como resultado la aplicación de un algoritmo goloso, sobre unos datos que fueron seleccionados mediante un algoritmo del mismo tipo. El algoritmo diseñado es eficiente porque su orden de complejidad es polinomial.

#### IV-C. Ejecución

La ejecución es mediante el Makefile correspondiente a Figura 2.

```

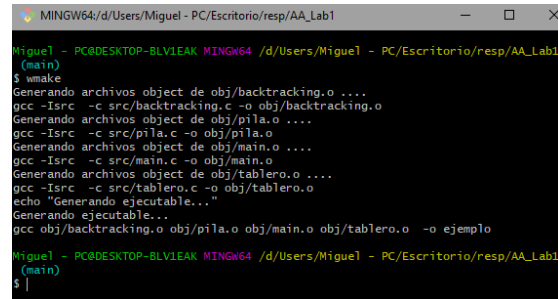
1 SRC:=src
2 OBJ:=obj
3
4
5
6 SOURCES := $(wildcard $(SRC)/*.c)
7 # Se obtiene los archivos .c de la carpeta en forma de
8 # lista de string
9
10 OBJECTS := $(subst $(SRC)/, $(OBJ)/, $(SOURCES))
11 # se cambia la lista de string por una lista de objetos
12 # que puede interpretarse make
13
14 PDBOY: clean all debug
15 # PDBOY se ocupa para decirle que al compilador que no
16 # es necesario crear los archivos clean all debug
17 # ya que estos son solo recetas
18
19 all: $(OBJECTS)
20     echo "Generando ejecutable..."
21     gcc -o $(OPTION) -o ejemplo
22
23
24 $(OBJ)/%.o: $(SRC)/%.c
25     echo "Generando archivos object de $@ ...."
26     gcc -c $(SRC) $(OPTION) -c $< -o $@
27
28
29 debug: OPTIONS := -g -O0
30
31 debug: all
32
33 # Una receta puede tener mas de una preparación
34 # make prepara las doc.
35
36 clean:
37     rm -rf $(OBJ)/*

```

Figura 2: Contenido del makefile

Para esto el programa se encuentra separado en 3 carpetas; incl: dónde se encuentran los archivos con .h, los cuales contienen la declaración de funciones a través de las cabeceras de las funciones junto con su respectiva documentación. src: dónde se encuentran los archivos .c, con la definición de las función y juntos a ellos el main.c. Y finalmente una carpeta llamada obj donde se albergan los archivos con la extensión .o que son creados posterior a la ejecución del makefile. Para la ejecución del makefile, se debe saber el comando a utilizar; para esto se debe ir al directorio bin (en linux) o path (en windows) e identificar el archivo que ejecutará el makefile (en windows el nombre por

defecto es mingw32-makefile, pero en el caso de las imágenes este último fue renombrado por wmake. Para linux el nombre es solo make). Como se observa en Figura 3.



```

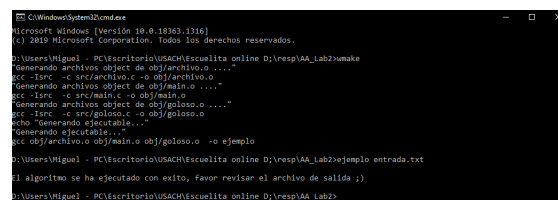
MINGW64/d/Users/Miguel - PC/Escritorio/resp/AA_Lab1
(main)
$ make
Generando archivos object de obj/backtracking.o ....
gcc -Isrc -c src/backtracking.c -o obj/backtracking.o
Generando archivos object de obj/pila.o ....
gcc -Isrc -c src/pila.c -o obj/pila.o
Generando archivos object de obj/main.o ....
gcc -Isrc -c src/main.c -o obj/main.o
Generando archivos object de obj/tablero.o ....
gcc -Isrc -c src/tablero.c -o obj/tablero.o
echo "Generando ejecutable..."
Generando ejecutable...
gcc obj/backtracking.o obj/pila.o obj/main.o obj/tablero.o -o ejemplo
MINGW64/d/Users/Miguel - PC/Escritorio/resp/AA_Lab1
(main)
$ |

```

Figura 3: Ejecución makefile

Al momento de la ejecución del makefile se tiene una opción importante que es poder ejecutarlo mediante el modo debug que para esto se debe agregar un debug a la ejecución del makefile (ejemplo windows: wmake debug) (ejemplo linux: make debug), esto habilitará la función printCurrent que irá mostrando la traza del algoritmo por consola. Es importante mencionar que cada vez que se compile de forma normal y se quiere compilar en modo debug es necesario usar el comando make clean o directamente borrar los archivos al interior de la carpeta obj o si no el cambio no se realiza, viceversa también debe realizarse tal proceso.

Una vez ejecutado el makefile se tendrá el programa compilado llamado ejemplo.exe, para la ejecución de este se debe de acompañar del nombre del archivo a leer, y posterior a su ejecución es que se escribirá un archivo llamado salida.out con el resultado del algoritmo. Como se observa en Figura 4. Para linux se debe ejecutar con el comando de



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.18363.1316]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Miguel> PC\Escritorio\USACH\Escuela online D:\resp\AA_Lab2\wmake
Generando archivos object de obj/archivo.o ....
gcc -Isrc -c src/archivo.c -o obj/archivo.o
Generando archivos object de obj/main.o ....
gcc -Isrc -c src/main.c -o obj/main.o
Generando archivos object de obj/goloso.o ....
gcc -Isrc -c src/goloso.c -o obj/goloso.o
echo "Generando ejecutable..."
Generando ejecutable...
gcc obj/archivo.o obj/main.o obj/goloso.o -o ejemplo
C:\Users\Miguel> PC\Escritorio\USACH\Escuela online D:\resp\AA_Lab2>ejemplo entrada.txt
El algoritmo se ha ejecutado con éxito, favor revisar el archivo de salida ;)
C:\Users\Miguel> PC\Escritorio\USACH\Escuela online D:\resp\AA_Lab2>

```

Figura 4: Ejecución programa compilado

la siguiente forma equivalente a la imagen: ./ejemplo mapal.txt

## V. CONCLUSIONES

Se logra la creación del software solicitado por Clover Sanity logrando un algoritmo que utiliza la estrategia de Goloso y la herramienta de Makefile resolviendo el problema para cualquier entrada que cumpla las condiciones de orden del archivo de entrada y de manera eficiente dado que el orden de complejidad es polinomial,  $n$  al cubo para ser exactos, por lo que es un algoritmo aceptable para entradas de gran flujo de datos.

Esta experiencia de laboratorio cierra adecuadamente la materia del método de Goloso con la aplicación realizada, es interesante comparar con otras estrategia, por ejemplo

Backtracking, a pesar de entregar el óptimo es comprensible una elección por Goloso dado que su es esquema es mas simple, intuitivo, aun así, desde la cosmovision de estudiantes de informática consideramos que si esta dentro de las posibilidades y en búsqueda de la mejor solución posible, siempre preferir Backtracking a pesar de su dificultad de implementación.