Algoritmos y estructuras de datos III

Trabajo Práctico Nº3

De Sousa Bispo Mariano	389/08	marian_sabianaa@hotmail.com
Grosso Daniel	694/08	dgrosso@gmail.com
Livorno Carla	424/08	carlalivorno@hotmail.com
Raffo Diego	423/08	enanodr@hotmail.com

Junio 2010

Índice

Introducción **2**

1. Situaciones de la vida real **2**

2. Algoritmo exacto **2**
2.1. Optimizacion 2
2.2. Complejidad temporal 2

3. Heurística constructiva **3**
3.1. Complejidad temporal 3

4. Busqueda local **4**
4.1. Complejidad temporal 4

5. Tabu-Search **5**
5.1. Complejidad temporal 5

6. Resultados **6**
6.1. Parametros de la heurística tabú 6
6.2. Comparacion de tiempos 6
6.3. Comparacion de calidad 6

7. Compilación y ejecución de los programas **6**

8. Conclusiones **6**

Introducción

Este trabajo tiene como objetivo la aplicación de diferentes técnicas algorítmicas para la resolución de tres problemas particulares, el cálculo de complejidad teórica en el peor caso de cada algoritmo implementado, y la posterior verificación empírica.

El lenguaje utilizado para implementar los algoritmos de todos los problemas fue C/C++

1. Situaciones de la vida real

2. Algoritmo exacto

2.1. Optimizacion

Dado que se trata de un algoritmo de backtracking, la optimzación se basa en podar las ramas en las que estamos seguros que no va a aparecer el óptimo. Para esto tenemos que poder predecir, dado un estado actual, si es posible mejorar el óptimo encontrado hasta el momento.

Por un lado, podamos las ramas que no forman un grafo completo, ya que no es solución.

Por otro lado, evaluamos en cada paso del algoritmo la cantidad de vértices que falta explorar. Es decir, calculamos el tamaño de la clique máxima que podriamos formar considerando los vértices que ya estan incluidos en la solución actual. Si la cantidad de vértices que todavía no fueron evaluados más la cantidad de vértices ya pertenecientes a la clique actual es menor a la cantidad de vértices de la clique máxima encontrada hasta el momento, no tiene sentido seguir explorando esa rama ya que el tamaño de la clique máxima que se puede encontrar por ese camino es menor al tamaño de la máxima encontrada. Por este motivo, podamos esta rama.

2.2. Complejidad temporal

3. Heurística constructiva

3.1. Complejidad temporal

4. Busqueda local

4.1. Complejidad temporal

5. Tabu-Search

5.1. Complejidad temporal

6. Resultados

6.1. Parametros de la heurística tabú

6.2. Comparacion de tiempos

6.3. Comparacion de calidad

7. Mediciones

- Para contar la cantidad aproximada de operaciones definimos una variable inicializada en *cero* la cual incrementamos luego de cada operación. Preferimos contar operaciones en vez de medir tiempo porque a pesar de que es aproximado el resultado, el error es siempre el mismo y así podemos hacer una mejor comparación entre las instancias.
  Midiendo tiempo, el error para cada instancia varía, ya que es el sistema operativo el que ejecuta nuestro programa, al "mismo tiempo"que otras tareas.

8. Compilación y ejecución de los programas

Para compilar los programas se puede usar el comando `make` (Requiere el compilador `g++`). Se pueden correr los programas de cada ejercicio ejecutando `./secuencia_unimodal`, `./ciudad` y `./prision` respectivamente.

Los programas leen la entrada de stdin y escriben la respuesta en stdout. Para leer la entrada de un archivo `Tp1EjX.in` y escribir la respuesta en un archivo `Tp1EjX.out` ses puede usar:
`./(ejecutable) <Tp1EjX.in >Tp1EjX.out`

Para contar la cantidad de operaciones: `./(ejecutable) count`. Devuelve para cada instancia el tamaño seguido de la cantidad de operaciones de cada instancia. En el ejercicio 3 también se puede contar la cantidad de operaciones en función de la cantidad de llaves de la siguiente manera:
`./prision count_llaves`. Devuelve para cada instancia la cantidad de llaves/puertas seguido de la cantidad de operaciones correspondientes.