Modelo Matemático para la Acomodación de un Basurero en una Ciudad.

Maria Andrea Cruz Blandón 0831816 Edgar Andrés Moncada Taborda 0832294
Hebert Vargas Tello 1124798 Luis Felipe Vargas Rojas 0836342

Enero 2013

Índice

1. Introducción **2**

2. Modelo Programación Lineal **2**
2.1. Datos de Entrada . . . 2
2.2. Variables del Problema . . . 2
2.3. Variables Binarias del Problema . . . 2
2.4. Restricciones Obvias . . . 3
2.5. Restricciones del Problema . . . 3
2.6. Función Objetivo. . . . 4

3. Branch and Bound **5**
3.1. Ejemplos tamaño de grilla 5 . . . 6
3.2. Ejemplos tamaño de grilla 10 . . . 9
3.3. Ejemplos tamaño de grilla 15 . . . 12
3.4. Ejemplos tamaño de grilla 20 . . . 15
3.5. Comparación todas las heurísticas usando piso . . . 18

4. Implementación **19**

5. Interfaz de Usuario **20**

6. Conclusiones **21**

1. Introducción

En el siguiente documento se explicará el modelo construido a través del paradigma de la programación linea (lp), para resolver el problema de la acomodación de un basurero en una ciudad, el problema se basa en que la construcción del basurero se debe realizar en el lugar más alejado posible de la ciudad más cercana, esto debido a los malos olores que genera y a las molestias de cada ciudad por dicha construcción.

Los principales problemas a los que nos enfentramos al tratar de modelar este problema con lp fueron definir los dominios las variables, establecer una forma para definir distancias entre ciudades y el basurero a través de un valor absoluto, definir a través de restricciones la ciudad más cercana entre otros detalles que se explicarán en el documento.

Para la implementación del modelo usamos el programa lpsolve y una librería de java que genera el archivo de entrada para el lpsolve teniendo en cuenta los valores de entrada.

2. Modelo Programación Lineal

2.1. Datos de Entrada

T	Tamaño de la grilla (T x T)
N	Numero de ciudades.
X_i	Coordenada x de la ciudad i.
Y_i	Coordenada y de la ciudad i.

2.2. Variables del Problema

Xb	Coordenada x del basurero
Yb	Coordenada y del basurero
Xc	Coordenada x de la ciudad más cercana al basurero
Yc	Coordenada y de la ciudad más cercana al basurero
Zx_i	Diferencia entre la coordenada x del basurero (Xb) y la coordenada x de la ciudad i
Zy_i	Diferencia entre la coordenada y del basurero (Yb) y la coordenada y de la ciudad i
Zx_{ia}	Usada para el valor absoluto de la diferencia entre entre la ciudad i y el basurero representa la parte positiva de la variable Zx_i
Zx_{ib}	Usada para el valor absoluto de la diferencia entre la ciudad i y el basurero, representa la parte negativa de la variable Zx_i
Zy_{ia}	Usada para el valor absoluto de la diferencia entre la ciudad i y el basurero representa la parte positiva de la variable Zy_i
Zy_{ib}	Usada para el valor absoluto de la diferencia entre la ciudad i y el basurero representa la parte negativa de la variable Zy_i
dx	Diferencia entre Xc - Xb *Usado para la función objetivo*
dy	Diferencia entre Yc - Yb *Usado para la función objetivo*
Zx	Valor absoluto de dx
Zy	Valor absoluto de dy

2.3. Variables Binarias del Problema

Usadas para el Valor Absoluto de la Diferencia de cada Ciudad con el Basurero:

Bx_{ia}	Usada en las restricciones del valor absoluto para la ciudad i coordenada x
Bx_{ib}	Usada en las restricciones del valor absoluto para la ciudad i coordenada x
By_{ia}	Usada en las restricciones del valor absoluto para la ciudad i coordenada y
By_{ib}	Usada en las restricciones del valor absoluto para la ciudad i coordenada y

Usadas para que $Xc \in X_i$ y $Yc \in Y_i$.

Bc_i Toma el valor de 1 si la ciudad i es la más cercana

Usadas para el valor absoluto de la función objetivo.

$Bx0$ Usada para el valor absoluto en la función objetivo coordenada x
$By0$ Usada para el valor absoluto en la función objetivo coordenada y

2.4. Restricciones Obvias

Dominios de las Variables

$$0 \leq Xb,Yb,Xc,Yc,\ Zx,\ Zy \leq T$$
Zx_i , Zy_i Son variables irrestrictas.

2.5. Restricciones del Problema

Garantizar que: $Xc \in X_i$ **y** $Yc \in Y_i$**.**

$$Xc = \sum_i X_i * Bc_i$$
$$Yc = \sum_i Y_i * Bc_i$$
$$\sum_i Bc_i = 1$$
$$i \in [1, N]$$

En la implementación se debe normalizar las restricciones por eso la sumatoria pasa al otro lado a restar y queda igual a cero la restricción, esto se hace con todas las restricciones.

Diferencias de distancias entre el basurero y la ciudad i, Zx_i**,**Zy_i

$$Zx_i = X_i - Xb$$
$$Zy_i = Y_i - Yb$$
$$i \in [1, N]$$

Valor absoluto para Zx_i**,**Zy_i

$$Zx_i = Zx_{ia} - Zx_{ib}$$
$$Zy_i = Zy_{ia} - Zy_{ib}$$

$$M * Bx_{ia} - Zx_{ia} >= 0$$
$$M * Bx_{ib} - Zx_{ib} >= 0$$

$$M * By_{ia} - Zy_{ia} >= 0$$
$$M * By_{ib} - Zy_{ib} >= 0$$

$$Bx_{ia} + Bx_{ib} = 1$$
$$By_{ia} + By_{ib} = 1$$

$$i \in [1, N]$$
$$M = T + 1$$

$$|Zx_i| = Zx_{ia} + Zx_{ib}$$
$$|Zy_i| = Zy_{ia} + Zy_{ib}$$

Para que Zx**,**Zy **tomen los valores de la ciudad más cercana al basurero se debe cumplir que:**

$$|Zx_i| + |Zy_i| \geq Zx + Zy$$
$$i \in [1, N]$$

Diferencia entre ciudad más cercana y basurero:

$$dx = Xc - Xb$$
$$dy = Yc - Yb$$

2.6. Función Objetivo.

Para la función objetivo nos encontramos de nuevo con el problema del valor absoluto ya que calculamos diferencias y podemos obtener resultados con valores negativos.

La función objetivo definida es:

$$\text{MAX}(\ Zx + Zy)$$

Donde Zx y Zy son la representación en valor absoluto de dx y dy, respectivamente por lo cual ésta función objetivo está sujeta a :

$$dx + M * B_{x0} - Zx \geq 0$$
$$dx + M * B_{x0} + Zx \leq M$$
$$dx \leq Zx$$
$$-dx \leq Zx$$
$$M = T * 2$$

$$dy + M * B_{y0} - Zy \geq 0$$
$$dy + M * B_{y0} + Zy \leq M$$
$$dy \leq Zy$$
$$-dy \leq Zy$$

3. Branch and Bound

Para la variables binarias que dan soporte a las restricciones relacionadas con los valores absolutos de las distancias y la función objetivo y que la ciudad más cercana corresponda a una ciudad del problema. Para ver mayor información revisar la sección **Variables Binarias del Problema**.

Para el algoritmo *Branch and Bound* (**B&B**) usamos la implementación en la librería *LpSolve*. Dicha librería trae varias heurísticas con las cuales usar el B&B, además de las heurísticas para elegir la rama a tomar, también esta disponible la opción de si iniciar la rama con la función techo o con la función piso.

De las heurísticas que están disponibles en *LpSolve* elegimos 13 para comparar el rendimiento. Además por cada heurística ésta se probó con la función techo y la función piso, para evaluar la mejor combinación. A continuación se explica el proceder de las 13 heurísticas[1]:

1. **NODE_FIRSTSELECT**: Elige la primera columna no entera.
2. **NODE_GAPSELECT**: Selecciona la variable de acuerdo a la distancia de los límites actuales.
3. **NODE_RANGESELECT**: Selecciona la variable de acuerdo al mayor límite actual.
4. **NODE_FRACTIONSELECT**: Selecciona la variable que tenga el valor fraccionario más grande.
5. **NODE_PSEUDOCOSTSELECT**: Selecciona la variable con la estrategia pseudo-costo (búsqueda costo uniforme)
6. **NODE_PSEUDONONINTSELECT**: Es una extensión de la estrategia pseudo-costo basada en minimizar el número de enteros fallidos.
7. **NODE_PSEUDORATIOSELECT**: También es una extensión de la estrategia pseudo-costo basada en maximizar la razón pseudo-cost divido por el número de fallidas. Similar a costo/beneficio.
8. **NODE_WEIGHTREVERSEMODE**: Selecciona la variable por el peor criterio en vez del mejor criterio.
9. **NODE_GREEDYMODE**: Búsqueda informada, con un costo heurístico.
10. **NODE_DEPTHFIRSTMODE**: Búsqueda en profundidad, selecciona el nodo por el que ya venia explorando.
11. **NODE_RANDOMIZEMODE**: Añade un factor random al costo de un nodo candidato.
12. **NODE_BREADTHFIRSTMODE**: Selecciona el nodo que no se halla seleccionado o que se halla seleccionado menos veces (anteriormente).
13. **NODE_AUTOORDER**: Crea una variable de ordenamiento "óptima" para el B&B. (Indexación)

Para realizar la pruebas correspondientes se generaron 20 ejemplos aleatorios. 5 ejemplos con una grilla de tamaño 5 y número de ciudades aleatorio, 5 ejemplos con una grilla de tamaño 10 y número de ciudades aleatorio, 5 ejemplo con una grilla de tamaño 15 y número de ciudades aleatorio y 5 ejemplos con una grilla de tamaño 20 y número de ciudades aleatorio. A cada ejemplo se le aplica el algoritmo con cada heurística una vez con la función techo y otra con la función piso.

Comparamos el desempeño de las heurísticas usando la función techo con los 5 ejemplos de un tamaño de grilla y así con todos los tamaños de la grilla. Se comparaban las heurísticas en 3 tres grupos, por cada grupo se seleccionaba la heurística con el mejor desempeño, y al final se eligió la mejor de las tres mejores. Este fue el procedimiento aplicado para las comparaciones por ejemplos. Finalmente se realizó una comparación entre todos los ejemplos (se eligió el un ejemplo por cada tamaño de grilla, el que tuviera mas ciudades) para elegir las 3 mejores heurísticas, dichas heurísticas usaban la función piso.

[1] fuente: http://lpsolve.sourceforge.net/5.5/set_bb_rule.htm

3.1. Ejemplos tamaño de grilla 5

Gráficas correspondientes al desempeño de las heurísticas, 3 grupos:
Usando techo:

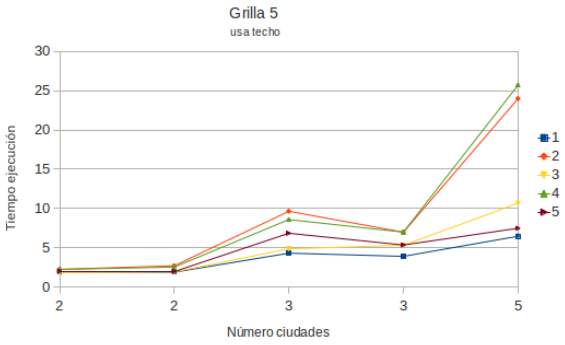

Figura 1: Comparación heurísticas 1 $\cdots$ 5

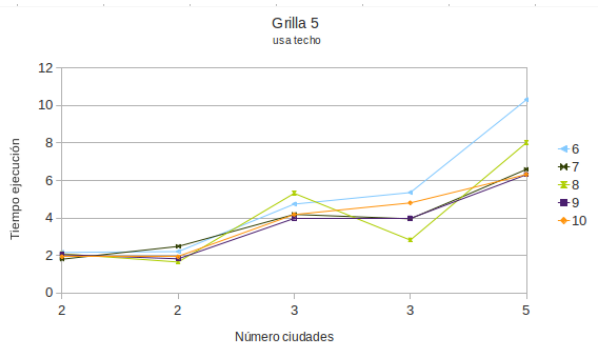

Figura 2: Comparación heurísticas 6 $\cdots$ 10

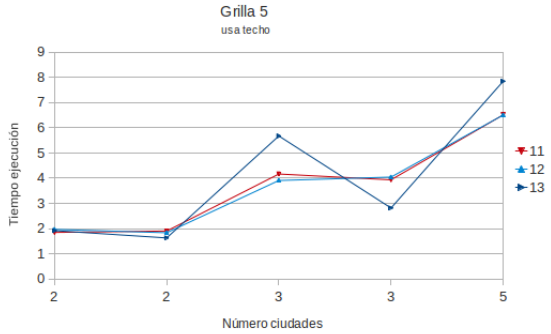

Figura 3: Comparación heurísticas 11 $\cdots$ 13

De la figura 1 la heurística que presentó mejor desempeño fue la número 1 (NODE_FIRSTSELECT), de la figura 2 fue la número 9 (NODE_GREEDYMODE) y de la figura 3 fue la número 12 (NODE_BREADTHFIRSTMODE). Al comparar éstas tres, la mejor correspondió a la heurística número 9 (NODE_GREEDYMODE).

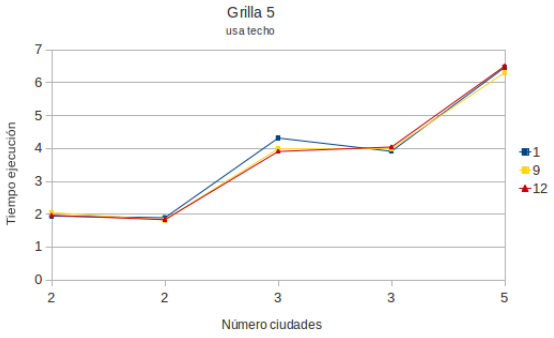

Figura 4: Comparación mejor heurística

Usando piso:

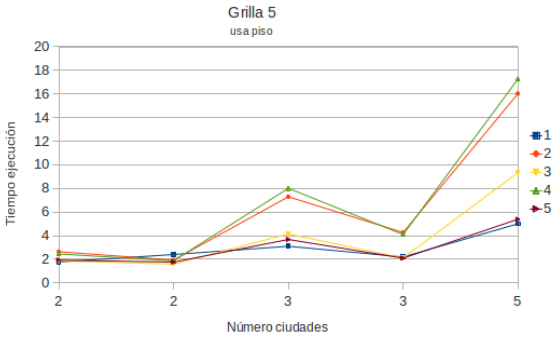

Figura 5: Comparación heurísticas 1 $\cdots$ 5

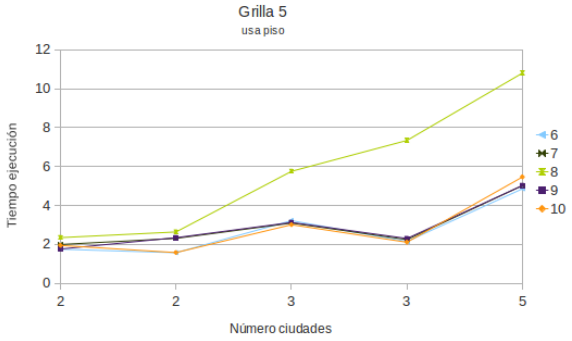

Figura 6: Comparación heurísticas 6 $\cdots$ 10

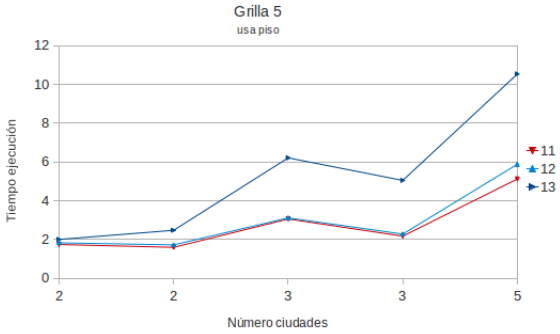

Figura 7: Comparación heurísticas 11 $\cdots$ 13

De la figura 5 la heurística que presentó mejor desempeño fue la número 1 (NODE_FIRSTSELECT), de la figura 6 fue la número 6 (NODE_PSEUDONONINTSELECT) y de la figura 7 fue la número 11 (NODE_RANDOMIZEMODE). Al comparar éstas tres, la mejor correspondió a la heurística número 6 (NODE_PSEUDONONINTSELECT).

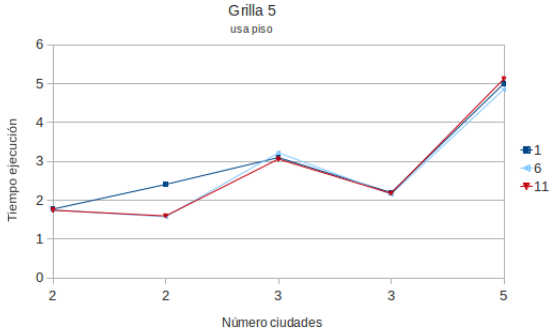

Figura 8: Comparación mejor heurística

Comparación mejor techo-piso

Tomamos la mejor heurística usando techo y la mejor heurística usando piso, comparamos sus resultados con los ejemplos y concluimos que la mejor era la número 6, que usaba piso.

6 (PISO) PSEUDONONINTSELECT	9 (TECHO) GREEDY
1,753	2,05
1,573	1,824
3,223	3,993
2,154	3,998
4,858	6,313

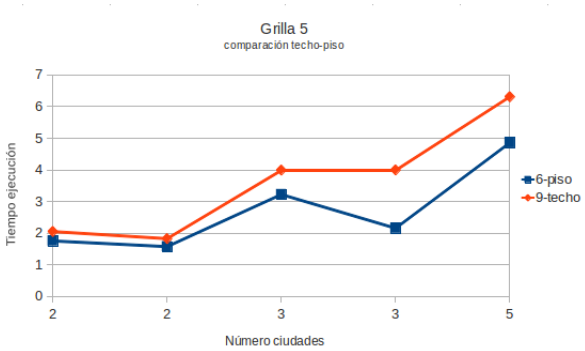

Figura 9: Comparación mejores techo y piso

3.2. Ejemplos tamaño de grilla 10

Gráficas correspondientes al desempeño de las heurísticas, 3 grupos:
Usando techo:

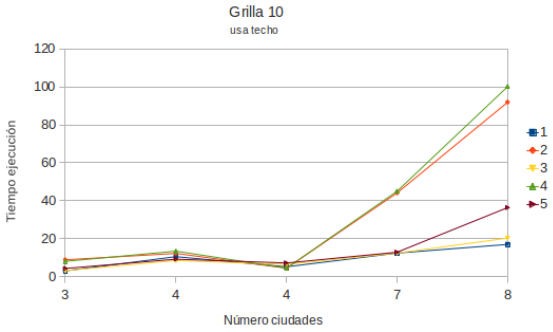

Figura 10: Comparación heurísticas 1 $\cdots$ 5

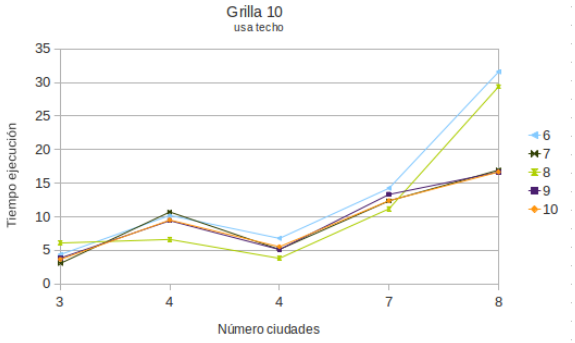

Figura 11: Comparación heurísticas 6 $\cdots$ 10

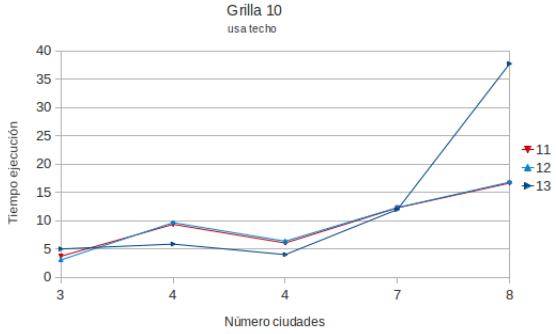

Figura 12: Comparación heurísticas 11 $\cdots$ 13

De la figura 10 la heurística que presentó mejor desempeño fue la número 1 (NODE_FIRSTSELECT), de la figura 11 fue la número 9 (NODE_GREEDYMODE) y de la figura 12 fue la número 11 (NODE_RANDOMIZEMODE). Al comparar éstas tres, la mejor correspondió a la heurística número 11 (NODE_RANDOMIZEMODE).

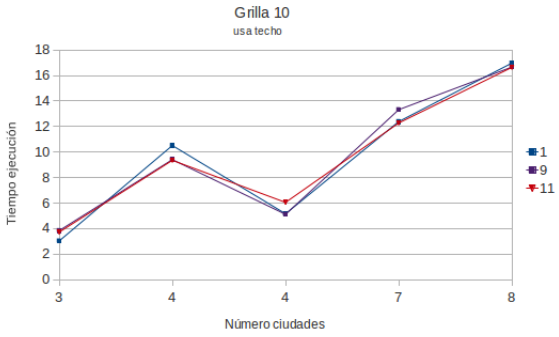

Figura 13: Comparación mejor heurística

Usando piso:

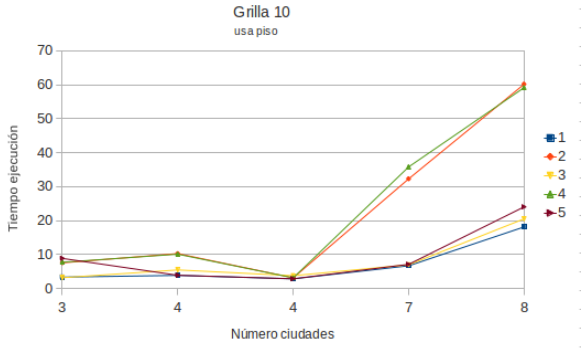

Figura 14: Comparación heurísticas 1 ··· 5

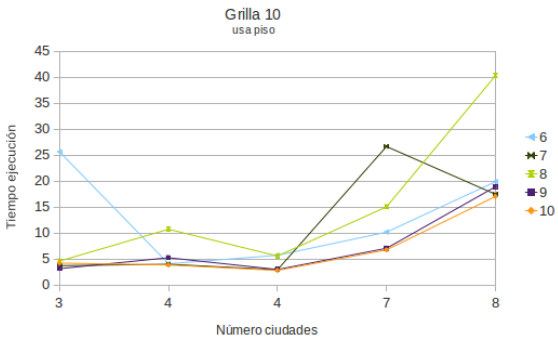

Figura 15: Comparación heurísticas 6 ··· 10

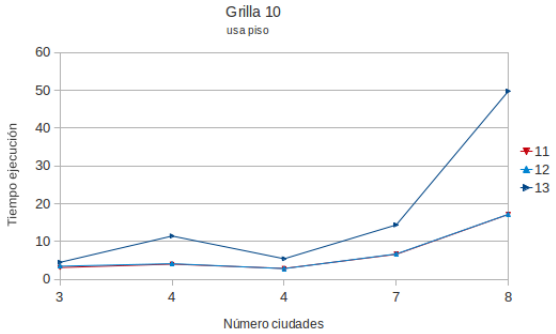

Figura 16: Comparación heurísticas 11 ··· 13

De la figura 14 la heurística que presentó mejor desempeño fue la número 1 (NODE_FIRSTSELECT), de la figura 15 fue la número 10 (NODE_DEPTHFIRSTMODE) y de la figura 16 fue la número 11 (NODE_RANDOMIZEMODE). Al comparar éstas tres, la mejor correspondió a la heurística número 10 (NODE_DEPTHFIRSTMODE).

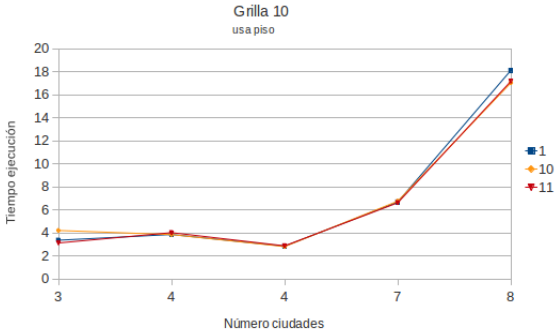

Figura 17: Comparación mejor heurística

Comparación mejor techo-piso

Tomamos la mejor heurística usando techo y la mejor heurística usando piso, comparamos sus resultados con los ejemplos y concluimos que la mejor era la número 10, que usaba piso.

10 (PISO) DEPTHFIRSTMODE	11 (TECHO) RANDOMIZEMODE
4,208	3,719
3,849	9,354
2,808	6,053
6,758	12,279
17,041	16,644

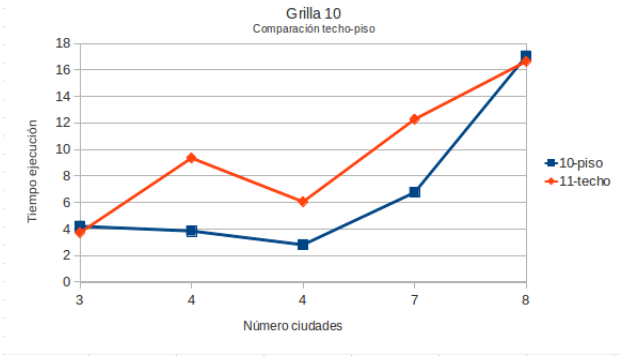

Figura 18: Comparación mejores techo y piso

3.3. Ejemplos tamaño de grilla 15

Gráficas correspondientes al desempeño de las heurísticas, 3 grupos:
Usando techo:

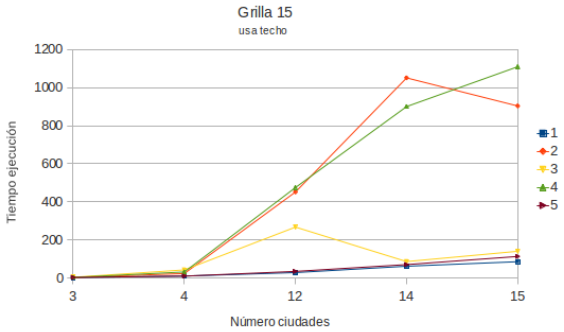

Figura 19: Comparación heurísticas 1 $\cdots$ 5

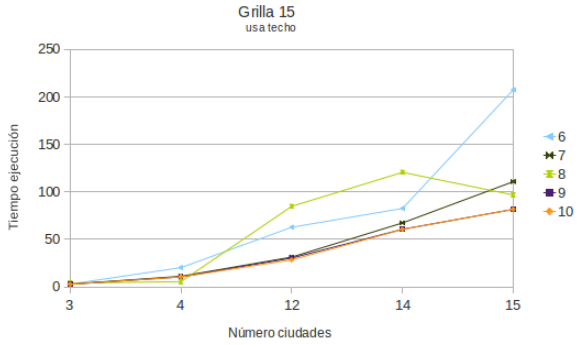

Figura 20: Comparación heurísticas 6 $\cdots$ 10

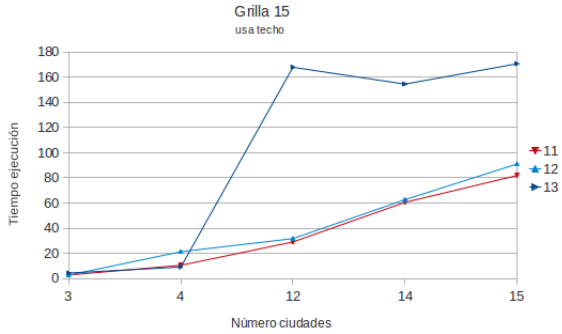

Figura 21: Comparación heurísticas 11 $\cdots$ 13

De la figura 19 la heurística que presentó mejor desempeño fue la número 1 (NODE_FIRSTSELECT), de la figura 20 fue la número 10 (NODE_DEPTHFIRSTMODE) y de la figura 21 fue la número 11 (NODE_RANDOMIZEMODE). Al comparar éstas tres, la mejor correspondió a la heurística número 10 (NODE_DEPTHFIRSTMODE).

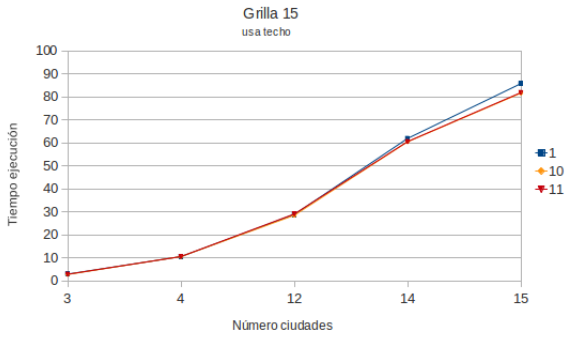

Figura 22: Comparación mejor heurística

Usando piso:

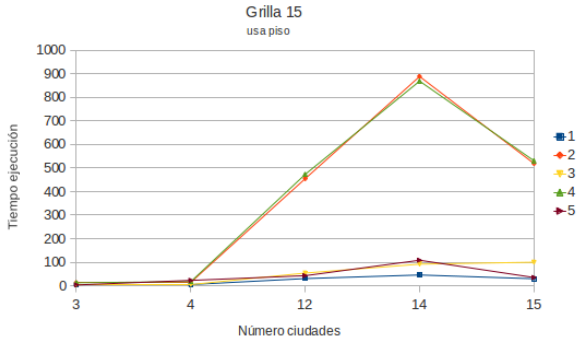

Figura 23: Comparación heurísticas 1 ··· 5

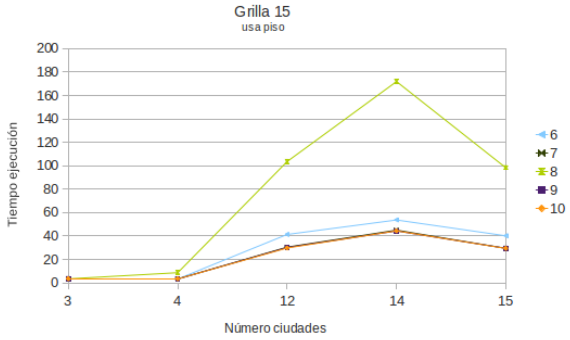

Figura 24: Comparación heurísticas 6 ··· 10

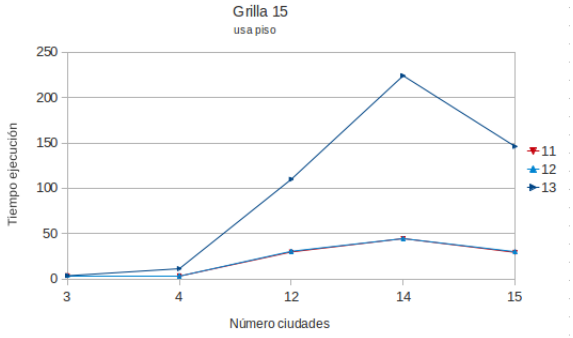

Figura 25: Comparación heurísticas 11 ··· 13

De la figura 23 la heurística que presentó mejor desempeño fue la número 1 (NODE_FIRSTSELECT), de la figura 24 fue la número 10 (NODE_DEPTHFIRSTMODE) y de la figura 25 fue la número 11 (NODE_RANDOMIZEMODE). Al comparar éstas tres, la mejor correspondió a la heurística número 10 (NODE_DEPTHFIRSTMODE).

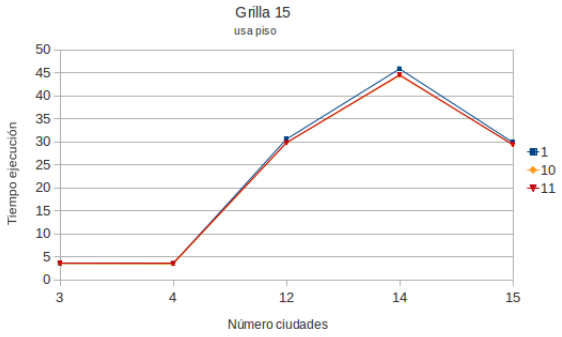

Figura 26: Comparación mejor heurística

Comparación mejor techo-piso

Tomamos la mejor heurística usando techo y la mejor heurística usando piso, comparamos sus resultados con los ejemplos y concluimos que la mejor era la número 10, que usaba piso.

10 (PISO) DEPTHFIRSTMODE	10 (TECHO) DEPTHFIRSTMODE
3,526	2,736
3,438	10,51
29,778	28,466
44,423	60,487
29,42	81,667

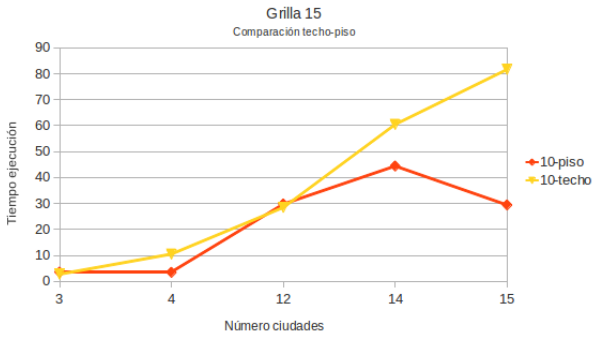

Figura 27: Comparación mejores techo y piso

3.4. Ejemplos tamaño de grilla 20

Gráficas correspondientes al desempeño de las heurísticas, 3 grupos:
Usando techo:

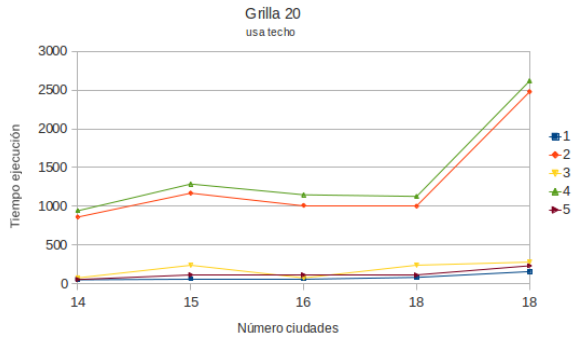

Figura 28: Comparación heurísticas 1 $\cdots$ 5

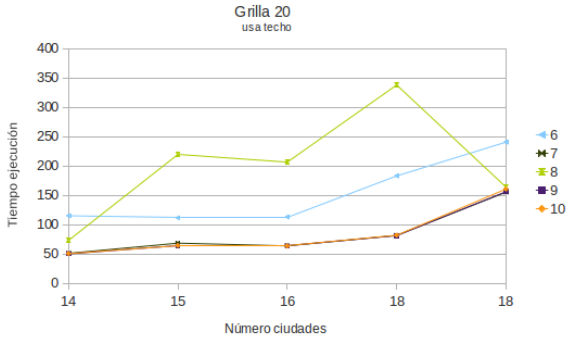

Figura 29: Comparación heurísticas 6 $\cdots$ 10

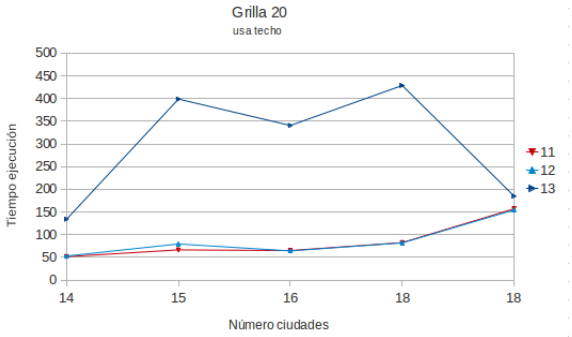

Figura 30: Comparación heurísticas 11 $\cdots$ 13

De la figura 28 la heurística que presentó mejor desempeño fue la número 1 (NODE_FIRSTSELECT), de la figura 29 fue la número 9 (NODE_GREEDYMODE) y de la figura 30 fue la número 12 (NODE_BREADTHFIRSTMODE). Al comparar éstas tres, la mejor correspondió a la heurística número 9 (NODE_GREEDYMODE).

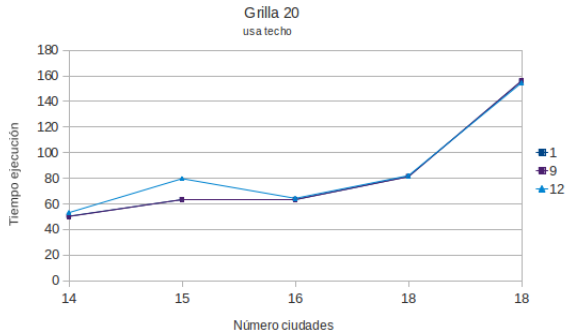

Figura 31: Comparación mejor heurística

Usando piso:

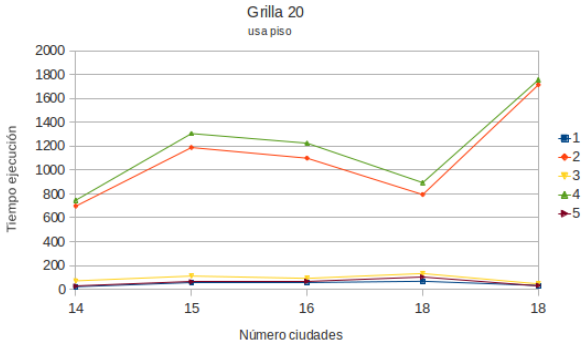

Figura 32: Comparación heurísticas 1 ··· 5

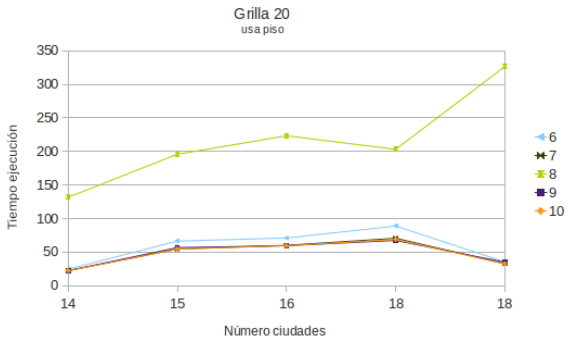

Figura 33: Comparación heurísticas 6 ··· 10

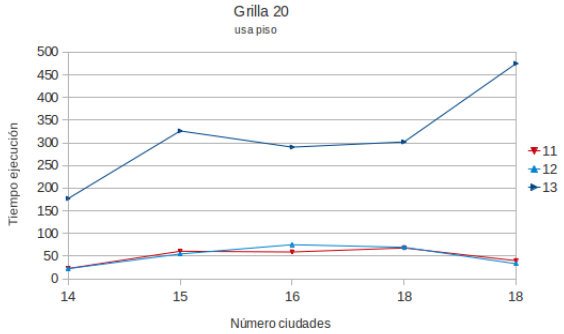

Figura 34: Comparación heurísticas 11 ··· 13

De la figura 32 la heurística que presentó mejor desempeño fue la número 1 (NODE_FIRSTSELECT), de la figura 33 fue la número 10 (NODE_DEPTHFIRSTMODE) y de la figura 34 fue la número 12 (NODE_BREADTHFIRSTMODE). Al comparar éstas tres, la mejor correspondió a la heurística número 1 (NODE_FIRSTSELECT).

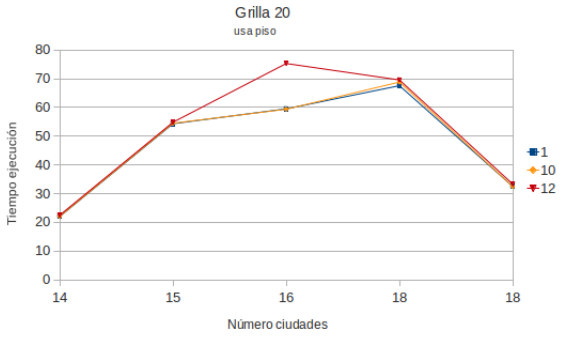

Figura 35: Comparación mejor heurística

Comparación mejor techo-piso

Tomamos la mejor heurística usando techo y la mejor heurística usando piso, comparamos sus resultados con los ejemplos y concluimos que la mejor era la número 1, que usaba piso.

1 (PISO) FIRSTSELECT	9 (TECHO) GREEDY
22,027	50,166
54,355	63,448
59,531	63,413
67,59	81,275
32,537	156,267

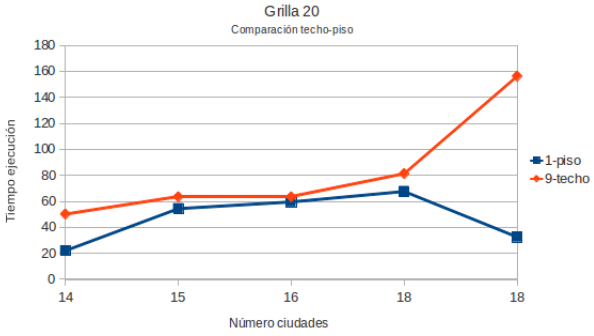

Figura 36: Comparación mejores techo y piso

3.5. Comparación todas las heurísticas usando piso

Como se evidencia en las gráficas anteriores, las heurísticas tuvieron mejor desempeño cuando usaban la función piso. Es por ello que para poder obtener las tres mejores heurísticas del B&B para nuestro modelo, comparamos todas las heurísticas usando la función piso con un ejemplo de cada una de las grillas (se eligió el ejemplo con mayor número de ciudades). Nuevamente graficamos el desempeño de las heurísticas en 3 grupos, seleccionamos las tres mejores y finalmente se plantea las conclusiones.

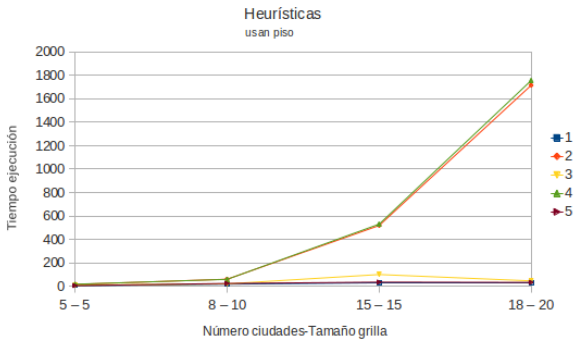

Figura 37: Comparación heurísticas $1 \cdots 5$

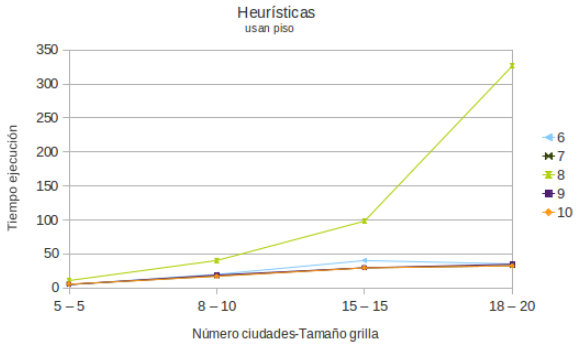

Figura 38: Comparación heurísticas $6 \cdots 10$

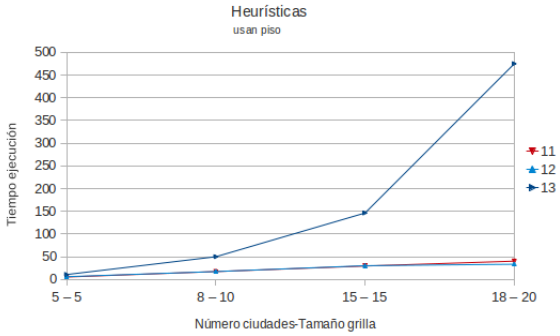

Figura 39: Comparación heurísticas $11 \cdots 13$

Finalmente comparamos la mejor heurística de cada grupo.

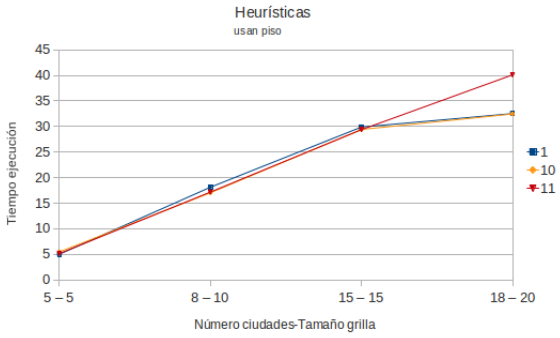

Figura 40: Comparación mejores heurísticas

Concluimos que la heurística que obtuvo un mejor desempeño fue la número 10 (NODE_DEPTHFIRSTMODE) y es la heurística que finalmente usamos para resolver el problema.

La función piso tuvo mejor comportamiento que la función techo en las heurísticas, y es por que nuestro modelo requería que solo una variable binaria de un conjunto tomará el valor de 1. Las variables binarias fueron usadas como flags de las restricciones. Por ello probar con que una variable tomara primero el valor de 0 resultó mas efectivo.

De las tres mejores heurísticas se destaca la opción de no perder tanto tiempo en que rama elegir sino en proceder con los subproblemas generados. Puesto que una vez se tiene activado la función piso, se debe de buscar de manera más rápida el valor de las otras variables no-enteras.

Heurísticas como la número 2, 4, 8 y 13 perdían mucho tiempo tratando de optimizar la variable a escoger, sin llegar rápido a la solución. Esto se debe a que debían realizar comparaciones de costos de cada una de las ramas para determinar por cual seguir.

4. Implementación

Para la implementación del modelo presentado anteriormente usamos el programa lpSolver, lpSolver es un programa para resolver problemas de programación lineal y programación entera.

LpSolver es un programa de código abierto licencia LGPL basado en el método simplex para resolver problemas de programación lineal y en el método branch and bound para resolver problemas de programación entera.

Usamos además la librería de LpSolver para java, puesto que éste fue nuestro lenguaje de programación. El proyecto fue desarrollado en Netbeans, se cuenta con una interfaz gráfica que permite ver la solución encontrada.

Un ejemplo de la implementación de una restricción es dado a continuación, lp corresponde al modelo en LpSolver:

```
1 LpSolve lp= LpSolve.makeLp(0, Ncol);
```

Función que instancia el lp se le define el numero de columnas (variables).

```
1 lp.addConstraintex(Ncol, row, colno, LpSolve.LE, sizeGrid);
```

Función que crea la restricción, row es un arreglo con los coeficientes de las variables ingresadas en el arreglo colno LpSolve.LE hace parte de las definiciones de las condiciones mayor igual, menor o igual, igual.

```
1 lp.setBinary(posVarBinary, true);
```

Se le indica al solver que la variable ubicada en la columna posVarBinary es binaria.

```
1
2 lp.setUnbounded(j + 1);
```

Se le indica al solver que la variable ubicada en la posición j+1 es irrestricta

```
1 lp.setColName(1, "Zx");
```

Función que nos permite asignarle un nombre a la variable para una mejor visualización.

```
1
```

```
lp.setObjFnex(Ncol, row, colno);
lp.setMaxim();
```

Definimos cuales son las variables involucradas en la función objetivo , luego definimos que debemos maximizar esa función

Para realizar las pruebas se realizó una clase que generara los archivos de prueba (.txt) y para obtener los datos de solución y desempeño se generaron archivos (.csv). Para ver estos archivos por favor revisar las carpetas (analisis_heuristicas) y (src/Solver/ejemplos_g_e) respectivamente.

5. Interfaz de Usuario

La imagen a continuación muestra la interfaz de usuario donde encontramos la grilla, donde se ubican las ciudades y el basurero, ademas en el panel derecho podemos ver información adicional como el valor de la función objetivo o las coordenadas del basurero

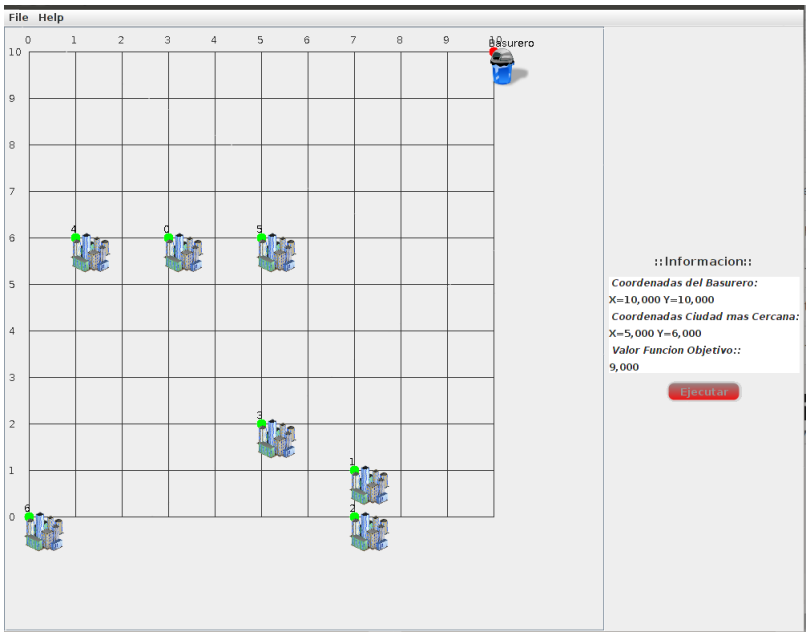

Figura 41: visualización en la interfaz de usuario

6. Conclusiones

- Al poder modelar un problema usando Programación Lineal se permite encontrar soluciones optimas de manera mas rapida que usando otros tipos de paradigma Sabiendo esto, la dificultad esta en plantear dicho modelo es por ello que la mayor inversión de tiempo en la programación LP se lleva acabo en el desarrollo del modelo matemático. Es este el que le da el soporte a las restricciones para que un problema pueda ser resuelto mas adelante con alguna implementación. Por lo cual un mal planteamiento puede que solo encuentre soluciones parciales que pueden ser buenas pero que no son las óptimas o por el contrario arroja resultados incorrectos para ciertos casos.
- Las restricciones *obvias* resultan de mucha importancia, pues acotan los espacios de búsqueda y establecen condiciones que de otra manera podrían ser violadas eventualmente. Un ejemplo de esto es la restricción que declara que la ciudad mas cercana (variable) debe pertenecer al conjunto de ciudades descritas en el problema.
- Diferentes heurísticas del B&B pueden darnos algunos resultados con variables con diferentes valores pero con la función objetivo igual, esto nos dice que conociendo bien el problema y las heurísticas a usar podríamos obtener un conjunto de soluciones óptimas para evaluar eventualmente cual sería una mejor solución para nuestro problema. Por ejemplo una ciudad cercana que colocara mas condiciones por construir el basurero tan cerca, de otra cuyas condiciones son menores, pero donde la función objetivo se mantiene.
- La heurística a elegir para el B&B depende en gran medida del modelo en particular. Pues una búsqueda que para un modelo resulte efectiva en otro puede ser mucho mas costosa y afectar los tiempo de respuesta. Es por ello que se debe analizar el modelo desarrollado y seleccionar las heurísticas mas adecuadas.
- La librería para java LpSolve permite implementar modelos de Programación Lineal con mayor fluidez. Esto ayuda a enfocarse mas en las restricciones que se están escribiendo que en cómo hacer restricciones en java. Enfocarse mas en el qué que en el cómo.