Informe Proyecto

Programación Por Restricciones

Planeación automática de horas de descargue para un puerto naval comercial

Profesor Juan Francisco Díaz

Michell Guzmán Cancimance

Cód. 0910016

Escuela de Ingeniería en sistemas y computación

Facultad de Ingeniería

Universidad del valle

Santiago de Cali

2009

Índice

1. Introducción

Objetivos

El proyecto tuvo como objetivo el de enfrentarnos a un problema de la vida real, a analizar dicho problema siguiendo el paradigma de restricciones, y la elaboración de un modelo que pudiera satisfacer las posibles restricciones que saldrían de nuestro modelo.

Alcance

En el proyecto se realizaron las restricciones basicas, tales como que todos los barcos sean atendidos, de lo contrario no habra solucion, que el estado de tiempo y de marea del muelle en un tiempo determinado, debe pertenecer al listado de tiempo y de marea de un barco determinado.

Tambien que los muelles deben atender un solo barco a la vez, y la restriccion de que se respete el tiempo de demora.

Tambien hay que marcar las restricciones que no se realizaron.

Las restricciones adicionales no se cumplieron, tampoco se realizo el control del tiempo de ejecucion, ni la recomputacion, etc.

2. Descripción del Problema

El problema es acerca de una actividad natural y frecuente en todo puerto, que consiste en la descarga de mercancía transportada por los barcos cargueros. Debido a la gran cantidad de barcos que atiende un determinado puerto comercial, al tiempo necesitado para descargar cada barco y al numero limitado de muelles de cada puerto, esta actividad debe ser programada con suficiente tiempo de anticipación para que no se presenten incrementos en los costos por demoras en la atención y descarga de los barcos ni inconvenientes graves como la perdida de mercancía por descargas a destiempo, o el encallamiento de un buque por bajas de marea o el deterioro de la carga por causa del clima, entre otros.

3. Modelo

Para la solución del problema el modelo que se utilizo fue el siguiente.

Lo que se busco fue generar una tupla de tuplas, que contendría la solución del problema, en donde cada tupla general me representa un barco que llega al puerto, mientras que cada subtupla contiene dos elementos, que me van a representar un muelle asignado a un barco y el tiempo de llegada al muelle de dicho barco, así como se muestra en la figura.

```
sol(sol(2 1)
    sol(1 1)
    sol(3 1)
    sol(4 1)
    sol(5 1)
```

Ej.: Sol (2 1), representa el barco 1, en el muelle 2 en la unidad de tiempo 1

Estructuras de datos

Se utilizaron estructuras de datos tales como, listas de tipos de datos simples, tuplas.

Parámetros de entrada

La información de entrada fue capturada desde un archivo con extensión .txt, formando una estructura como la que se muestra a continuación.

Barcos (Barco (Código: 1 Descripción: descripción (TiempoMax: 3 EstadoTiempo: (1 3) Demora: 3) EstadoMarea: (1 2))

Barco (Código: 2 Descripción: descripción (TiempoMax: 3 EstadoTiempo: (1 3) Demora: 3) EstadoMarea:(1 2)))

En donde:

Barcos representa una tupla de barcos, que a su vez representa una tupla con 3 valores.

Descripción representa una tupla con 3 valores

Estado de tiempo y estado de marea representan listas de estados de tiempo, tomando:

1: Seco

2: Húmedo

3: Lluvia

Y el estado de marea

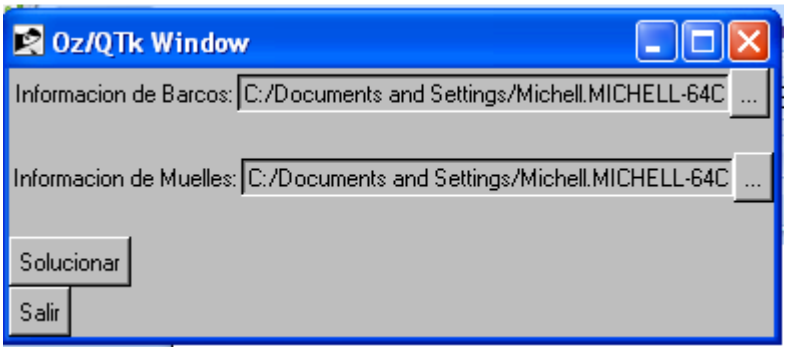
1: Baja

2: Media

3: Alta

Para la carga de la información se hizo uso de dos archivos con extensión .txt, los cuales contienen la información de entrada de los barcos que arribaran al puerto, y la información de los muelles de dicho puerto.

Los archivos de texto son cargado mediante una interfaz grafica simple, como la que se muestra en la siguiente figura.



Restricciones

Las restricciones básicas a satisfacer son las siguientes:

- Todos los barcos que arriban al puerto deben ser atendidos, de lo contrario no existiría solución para el problema.

  Esta restricción se define por defecto al asociar al dominio de muelles de la tupla solución el dominio de 1::M, donde M representa el numero de muelles que hay en el barco, con lo que se obligaría a tomar un valor de 1 a M.

```
for I in 1..N do
   Solucion.I = {MakeTuple sol 2}
   Solucion.I.1::1#M  % Todos los barcos son atendidos
   Solucion.I.2::1#H
end
```

- Cada muelle solo puede atender un barco a la vez

En esta restricción lo que se hizo fue preguntar si el tiempo de llegada al muelle de mi barco es igual al tiempo de llegada de otro barco del puerto, entonces mi muelle debe ser diferente.

```
thread
 for I in 1..N do
    for J in 1..N do
       if {Bool.and (Solucion.I.2 == Solucion.J.2) (I \= J)} then
          Solucion.I.1 \=: Solucion.J.1
       end
    end
 end
end
```

- Se debe respetar el tiempo máximo de espera definido por la carga

Esta restricción queda cobijada por las siguientes restricciones.

- Durante el tiempo de espera y descarga de un barco, el estado de tiempo debe estar dentro de la lista de estados posibles de tiempo definidos por el tipo de carga

Esta restricción se cumplió comparando si en el rango de tiempo desde que llegue hasta la suma del tiempo de llegada, más el tiempo máximo de espera, más el tiempo de descarga, el estado de tiempos del muelle coincidiera con la lista de estados de tiempo posibles del barco a atender.

```
thread
  for I in 1..N do
     for J in 1..M do
        for K in (Solucion.I.2)..(Barcos.I.2.1 + Barcos.I.2.3 + Solucion.I.2) do
          {List.member {List.nth Muelles.J.2 K} Barcos.I.2.2 true}
          {List.member {List.nth Muelles.J.3 K} Barcos.I.3 true}
        end
     end
  end
end
```

- Durante el tiempo de espera y descarga de un barco el estado de marea debe estar dentro de la lista de posibles estados de marea definida por el barco.

Ídem al punto anterior.

- Restriccion utiizada para garantizar que el tiempo de demora de descarga de una barco se respete

```
thread
  for I in 1..N do
     for J in 1..N do
        if {Bool.and {Solucion.I.1 == Solucion.J.1} {I \= J}} then
           {FD.disjoint Solucion.I.2 Barcos.I.2.3 Solucion.J.2 Barcos.J.2.3}
        end
     end
  end
end
```

Estrategias de distribución

Debido a que en la solución de mi problema se debe asignar un tiempo y un muelle a un barco en particular, entonces se debe distribuir tanto por muelles como por barcos.

Básicamente mi estrategia de distribución es parecida al naive en donde se busca elegir para la distribución a la variable, en este caso el muelle que contenga el menor dominio, y esto se ajusta debido a que en las restricciones implementadas lo que se busca es podar el dominio de un barco en particular.

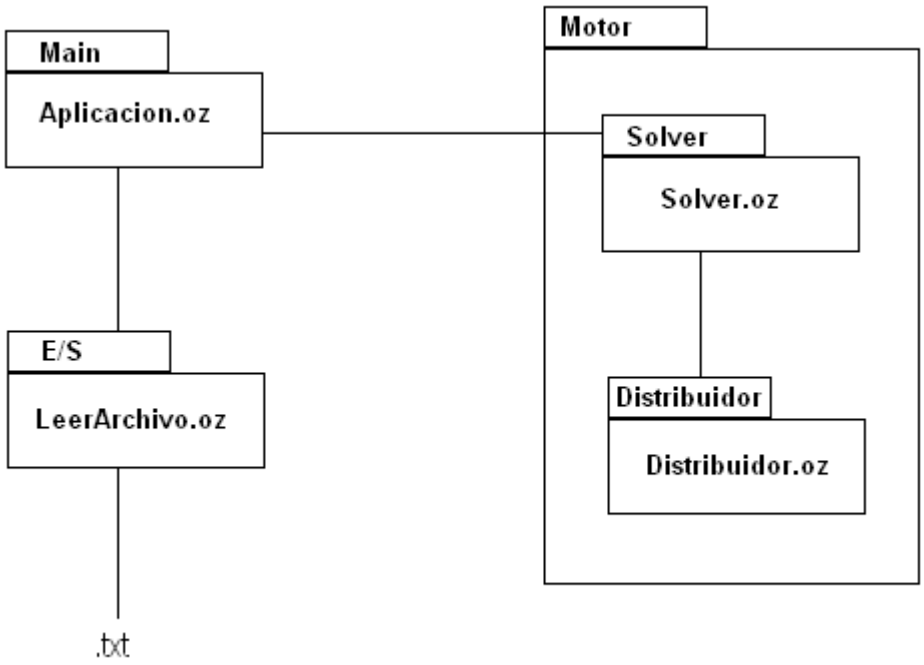
La estrategia de distribución es:

```
proc {Distribuidor1 ListaMuelles}
{Space.waitStable}
local
   Fs={Filter ListaMuelles fun {$ I} {FD.reflect.size I}>1 end}
in
   case Fs
   of nil then skip
   [] F|Fr then M={FD.reflect.min F} in
      choice F=M   {Distribuidor1 Fr}
      []     F\=:M {Distribuidor1 Fs}
      end
   end
end
end
```

En donde Fs representa los dominios de la lista de muelles que todavía no son valuación.

Y M representa el menor dominio de Fs

Arquitectura De la Aplicación


Main
Aplicacion.oz
Motor
Solver
Solver.oz
E/S
LeerArchivo.oz
Distribuidor
Distribuidor.oz
.txt

4. Pruebas

En todas las siguientes pruebas solo se busca una solucion

Archivo de prueba 1 (Buscando una solucion)

Para este ejemplo se tiene 1 muelle con 2 barcos, con 20 unidades de tiempo, los estados de tiempo y de marea de los barcos tienen todos los posibles estados, es decir:

Estados de tiempo de los barcos

1: Seco

2: Húmedo

3: Lluvia

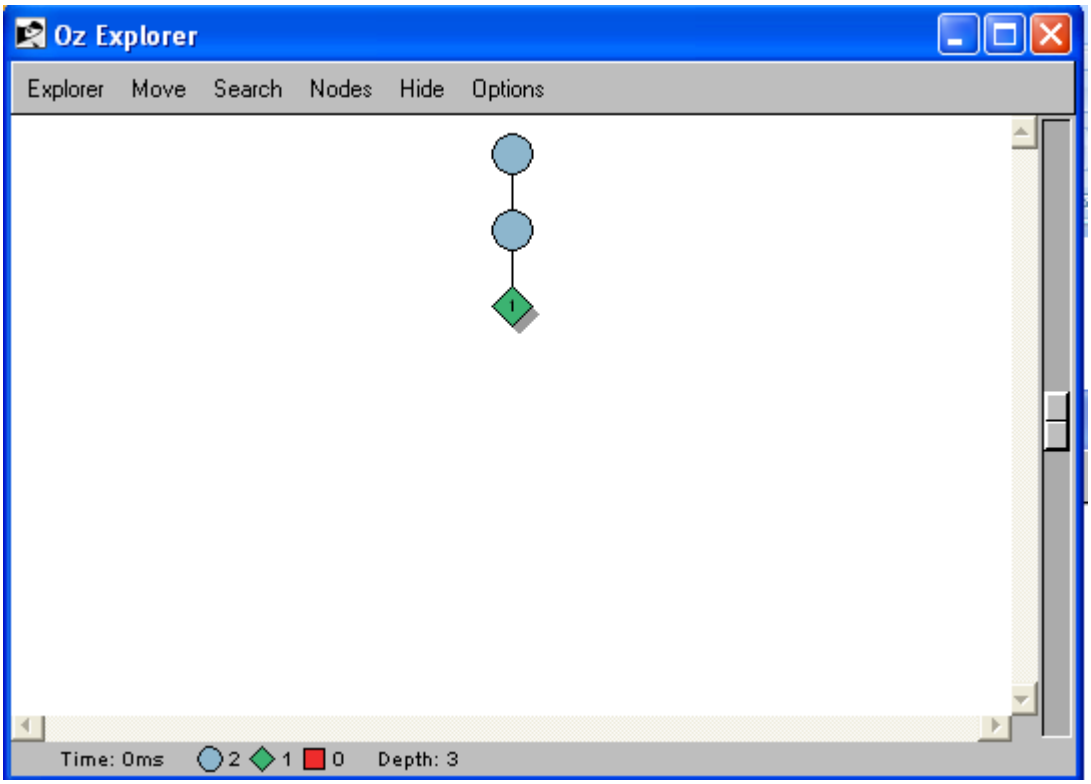
Y el estado de marea

1: Baja

2: Media

3: Alta

El arbol de busqueda generado por la aplicación, usando una distribucion que toma la variable con el menor dominio es:



Y la solucion que genera es

Solucion

```
1#
solucion(barco(1 1)
         barco(1 6))
```

En donde barco(muelle:1 hora:1)

Archivo de prueba 2(Buscando una solucion)

Para este ejemplo se tienen 20 unidades de tiempo, con 5 muelles y 7 barcos, los estados de tiempo y de marea de los barcos tienen todos los posibles estados, es decir:

Estados de tiempo de los barcos

1: Seco

2: Húmedo

3: Lluvia

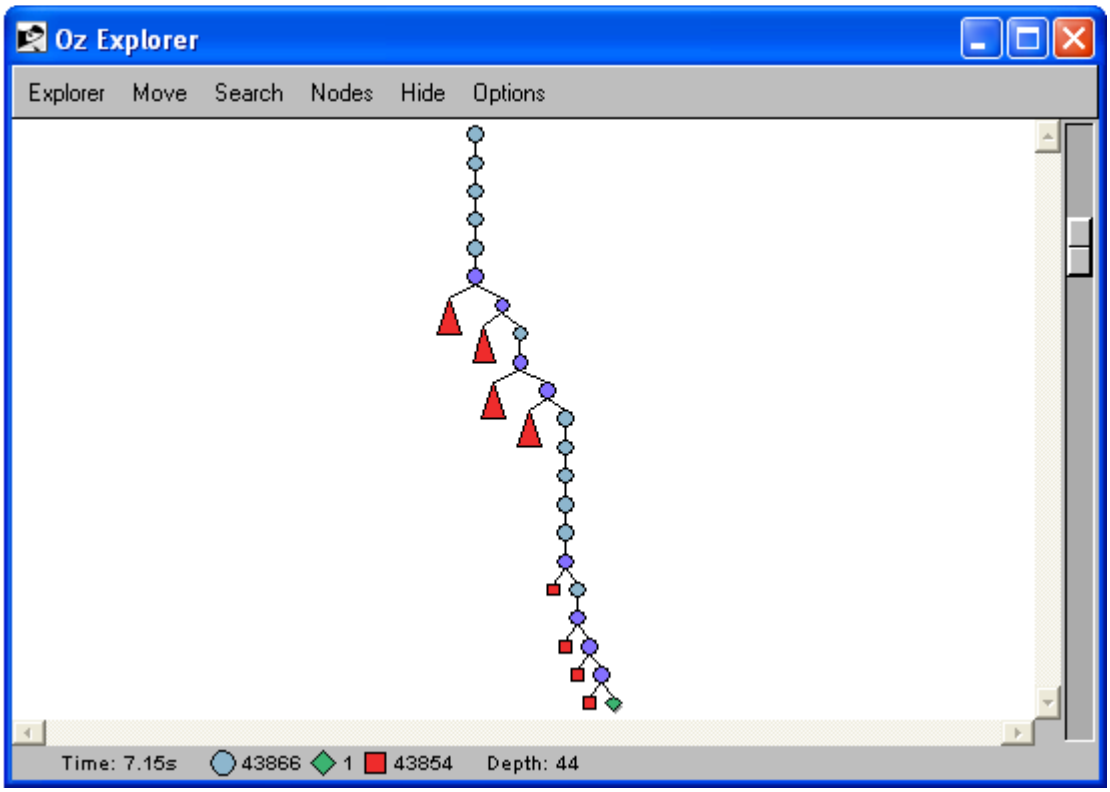
Y el estado de marea

1: Baja

2: Media

3: Alta

El arbol de busqueda generado por la aplicación, usando una distribucion que toma la variable con el menor dominio es:



Y la solucion que genera es

Solucion

```
1#
solucion(barco(1 1)
         barco(2 1)
         barco(3 1)
         barco(4 1)
         barco(5 1)
         barco(2 3)
         barco(5 3))
```

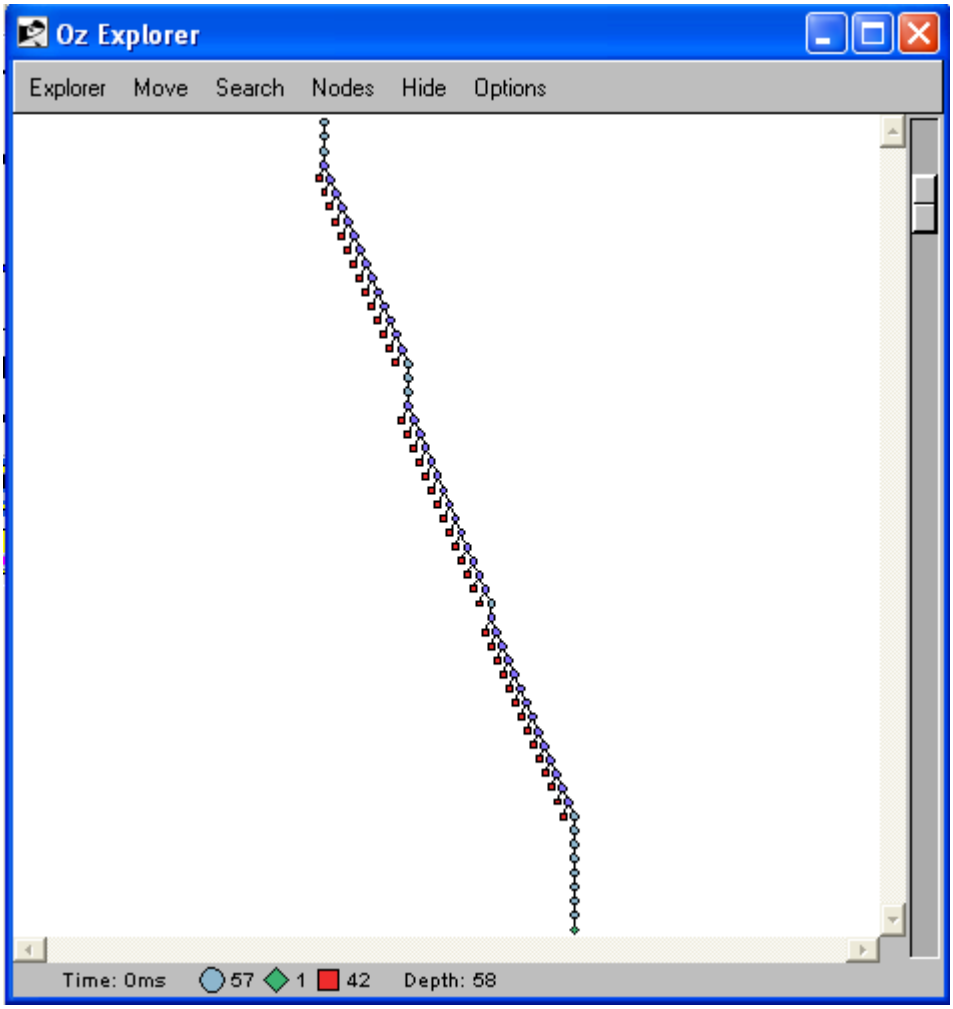
En donde barco(muelle:1 hora:1)

Archivo de Prueba 3 (Buscando una solucion)

Para este ejemplo se tienen 20 unidades de tiempo, con 5 muelles y 8 barcos, los estados de tiempo y de marea de los barcos no tienen todos los posibles estados, es decir:

Algunos tienen dos estados y otros tienen los tres tanto de tiempo como de marea.

El arbol de busqueda generado por la aplicación, usando una distribucion que toma la variable con el menor dominio es:



Y la solucion que genera es

Solucion

```
1#
solucion(barco(1 1)
         barco(2 1)
         barco(3 1)
         barco(1 15)
         barco(4 1)
         barco(5 1)
         barco(2 15)
         barco(3 15))
```

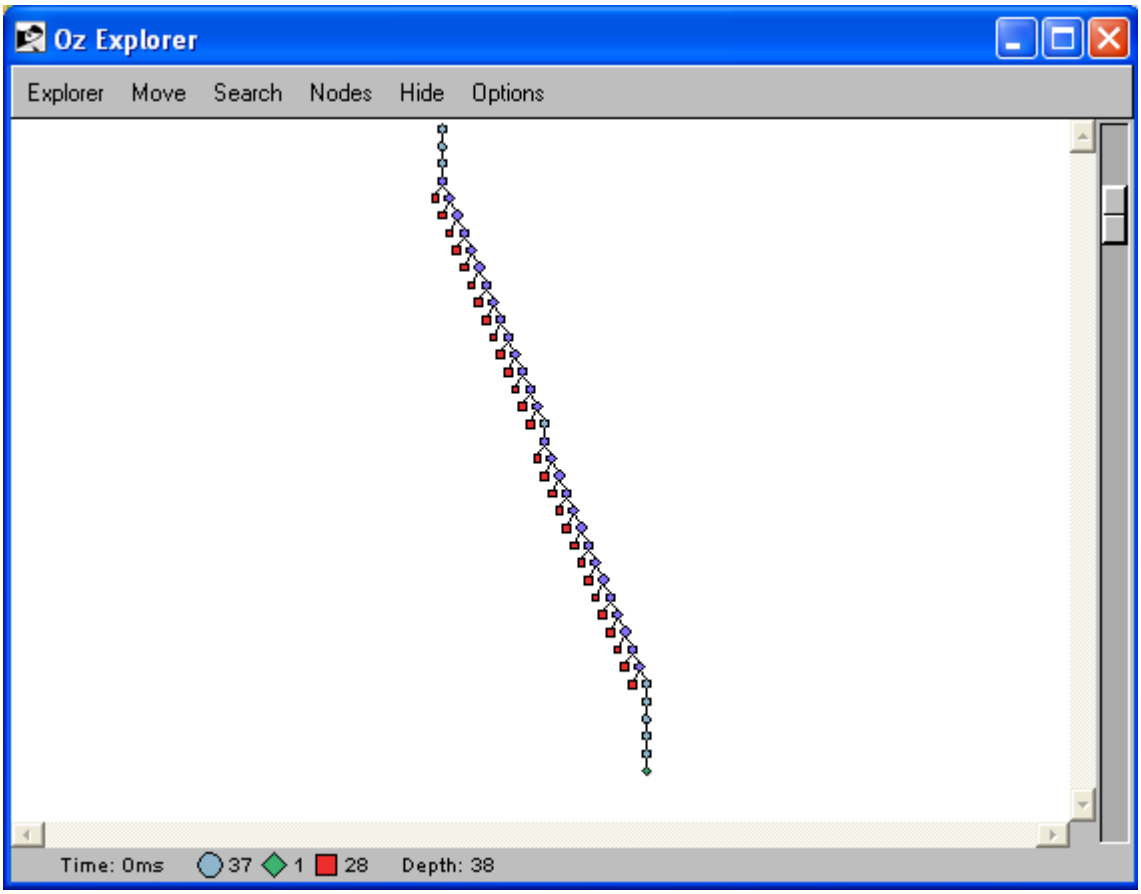
En donde barco(muelle:1 hora:1)

Archivo de prueba 4 (Buscando una solucion)

Para este ejemplo se tienen 20 unidades de tiempo, con 3 muelles y 5 barcos, los estados de tiempo y de marea de los barcos no tienen todos los posibles estados, es decir:

Algunos tienen dos estados y otros tienen los tres tanto de tiempo como de marea.

El arbol de busqueda generado por la aplicación, usando una distribucion que toma la variable con el menor dominio es:



Y la solucion que genera es

Solucion

```
1#
solucion(barco(1 1)
         barco(2 1)
         barco(3 1)
         barco(1 15)
         barco(2 15))
```

En donde barco(muelle:1 hora:1)

Archivo de prueba 5 (Buscando una solucion)

Para este ejemplo se tienen 20 unidades de tiempo, con 1 muelle y 5 barcos, los estados de tiempo y de marea de los barcos tienen todos los posibles estados, es decir:

Estados de tiempo de los barcos

1: Seco

2: Húmedo

3: Lluvia

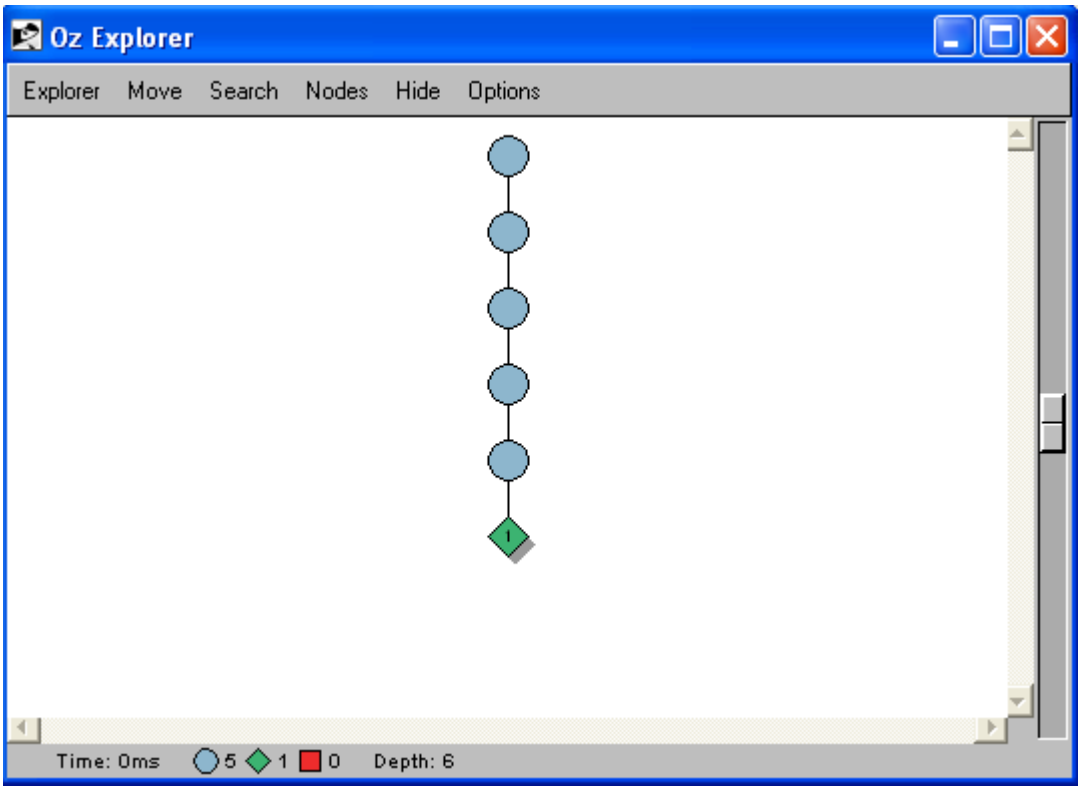
Y el estado de marea

1: Baja

2: Media

3: Alta

El arbol de busqueda generado por la aplicación, usando una distribucion que toma la variable con el menor dominio es:



Y la solucion que genera es

Solucion

```
1#
solucion(barco(1 1)
         barco(1 4)
         barco(1 6)
         barco(1 10)
         barco(1 13))
```

En donde barco(muelle:1 hora:1)

5. Problemas Presentados

En un principio el principal problema que tuve que afrontar fue el de adaptarme al paradigma de restricciones, y al estilo de programación en oz.

En la lectura del archivo y la interfaz grafica no se presentaron problemas tan graves, debido a la ayuda de la aplicación de ejemplo.

Luego pensar en el modelo fue difícil en un principio, pero luego analizando los datos de entrada y el objetivo de la aplicación, entonces se fue haciendo menos engorroso.

Después de tener claro como se iba a modelar el problema se procedió a satisfacer las restricciones básicas de la aplicación, en donde se presentaron los problemas mas graves.

El primer problema fue cuando se trato de definir un dominio para cada uno de los valores de la solución que se planteo, definir un dominio para los muelles no fue problema, pero para las horas si porque no sabia como debia ser distribuido el tiempo, luego me di cuenta de que cada unidad de tiempo equivalía a 5 minutos y pude solucionar el problema.

Luego la restricción de que un muelle solo puede atender un barco a la vez se tuvo que realizar un poco más de análisis, y luego se llego a la conclusión de que si dos o más barcos tienen el mismo tiempo de llegada, entonces el muelle para esos barcos debe ser diferente.

El problema mas grave y mas difícil de solucionar fue el de lograr que si dos barcos eran atendidos en el mismo muelle, entonces el tiempo de llegada del barco 2 debería respetar el tiempo atención del barco 1, finalmente este problema se logro resolver usando disjoint, en donde si dos barcos comparten el mismo muelle, el tiempo de llegada del barco 1, sumado a su demora, debe ser menor a la suma de la llegada del barco 2, sumado con su demora, de esta manera fue posible solucionar el problema.

Se tuvo algunos problemas con el tiempo de espera de cada barco, es decir el barco llega a la hora que va a ser atendido.

6. Conclusiones

El objetivo del proyecto que fue el de enfrentar al estudiante a como modelar un problema de la vida real como un csp, se cumplio, logro hacer que se cambiara la forma de pensar al enfrentarnos a un problema de este tipo, y visualizarlos desde el punto de vista del paradigma de restricciones.

Se logro interactuar con la programacion de alto nivel, en la cual nos podemos olvidar de cómo el motor de mozart soluciona el problema, y nos permite enfocarnos plenamente en el modelamiento del problema, claro que al principio se hizo un poco complicado acostumbrase a este estilo de programacion, y en especial al paradigma de restricciones.

Viendo de esta manera el no determinismo no observable, es decir, que no nos importa en que orden se ejecuta nustra aplicación, debido a que siempre nos retorna el resultado esperado.

7. Bibliografia

- Priciples of constraint programming - Apt
- Conceptos Técnicas y Modelos de Programación – Peter Van Roy, Seif Haridi
- Pagina web de mozart-oz http://www.mozart-oz.org