Universidad Central “Marta Abreu” de Las Villas

Facultad de Matemática, Física y Computación

TESIS DE MAESTRIA EN COMPUTACION APLICADA

Incorporación de nuevas funcionalidades al lenguaje de programación TOL.

Autor: Liván Daniel Ramírez Dorta

Tutor: Dr. Daniel Gálvez Lío

“Año 49 de la Revolución”
Santa Clara, 2007

Resumen

Hoy en día diversas instituciones requieren conocer el comportamiento futuro de ciertos fenómenos con el fin de planificar, prever o prevenir. El análisis de series temporales se ha convertido en la técnica más importante en esta área, dentro de los lenguajes de programación existentes para este fin se encuentra *TOL*. El presente trabajo trata sobre la incorporación a este lenguaje de un conjunto de funciones y herramientas que lo convierten en un instrumento más potente para la modelación estadística. Se proponen e implementan métodos para la compactación de los datos que utiliza y se describe la incorporación de un nuevo modelo de estimación con estructura jerárquica y orientado a manipular grandes volúmenes de información.

Indice

Introducción

Toda institución, ya sea la familia, la empresa o el gobierno, tiene que hacer planes para el futuro si ha de sobrevivir y progresar. La previsión, a su vez, se suele basar en lo que ha ocurrido en el pasado. Se tiene pues un nuevo tipo de inferencia estadística que se hace acerca del futuro de alguna variable o compuesto de variables basándose en sucesos pasados [ARE01], [CHA75]. La técnica más importante para hacer inferencias sobre el futuro con base en lo ocurrido en el pasado, es el análisis de series de tiempo.

Orientado a este objetivo, fue creado en 1997 el lenguaje de programación TOL, un lenguaje dedicado al mundo de la estadística y principalmente enfocado al análisis de series temporales y procesos estocásticos [BOX94], [HAM94]. Su nombre proviene de las letras iniciales de las palabras en inglés Time Oriented Language, que en español es lenguaje de programación orientado al tiempo.

Bayes Inference SA [http://www.bayesforecast.com/] lo usa como herramienta para el análisis estadístico de datos, dando así servicio a sus clientes, entre los que hay empresas distribuidoras de periódicos, compañías telefónicas, agencias de publicidad, etc. Las soluciones que Bayes Forecast desarrolla permiten entender el pasado, controlar el presente, prever los probables resultados futuros y tomar continuamente las mejores decisiones.

El lenguaje TOL debe por tanto ser capaz de manejar millones de datos, todos los datos disponibles para el análisis del mercado correspondiente de nuestro cliente de forma eficaz y eficiente, desarrollando respuestas automáticas de previsión [MAK83]. El tipo de problemas que se resuelve usando TOL es muy variado.

Por ejemplo para compañías telefónicas se estudian los datos de uso de las líneas, analizando el perfil del cliente con el objetivo de maximizar su uso. En este caso, los datos a analizar para poder modelar los fenómenos que se pretende explicar y posteriormente predecir, son de gran magnitud, sólo para tener una idea baste decir que por cada llamada telefónica efectuada se introduce un nuevo registro en las tablas de datos, por lo que se hacen necesario mecanismos que permitan compactar estos datos con el objetivo de optimizar espacio, pero no sólo eso, esta compactación debe realizarse de manera tal que el acceso a la misma sea de forma aleatoria, y por supuesto realizar esto con una velocidad adecuada.

Para una empresa distribuidora de periódicos se analiza la venta y la distribución de cada tirada, se estudia la realidad diaria y como se comporta la venta día a día en función de lo que ocurre en la vida diaria, se establecen modelos estadísticos que describen esos

sucesos y de su análisis se establece la venta futura. Para esto se hace necesario contar con métodos de estimación, donde se aproveche la información que aportan cada uno de los kioscos y a la vez la relación existente entre ellos, el problema aquí radica en lo masivo que deben ser estos métodos pues se habla de considerar en un mismo modelo por ejemplo todos los kioscos que distribuyen un periódico determinado en todo un país.

Surge entonces la interrogante:

¿Se podrá dotar a TOL de mecanismos para la compactación de los datos que debe analizar e introducir un nuevo esquema de modelación para mejorar la estimación actual?

El presente trabajo tiene como objetivo general:

Proponer, fundamentar e implementar métodos para la compactación de los datos que utiliza TOL e incorporarle un nuevo esquema de modelación.

Son objetivos específicos:

- Proponer la estrategia y los métodos adecuados para lograr la compactación de los datos que utiliza TOL.
- Revisar y asimilar las implementaciones existentes de bibliotecas de estructuras de datos y algoritmos en memoria externa que se puedan usar.
- Implementar el compactador y verificar los resultados obtenidos de la aplicación del mismo a los datos que se necesitan procesar.
- Incorporar funciones de clasificación que permitan el proceso de construcción de la estructura jerárquica del problema a modelar.
- Describir la estructura computacional para representar un modelo jerárquico lineal y los algoritmos para su manipulación.
- Construir herramientas que soporten la arquitectura informática necesaria en la resolución de este tipo de modelo masivo.

En el análisis de la información de las compañías telefónicas, la flexibilidad con la que se accede a los datos y los tiempos de recuperación son muy importantes, teniendo un instrumento de almacenamiento de la información compactada que facilita su recuperación rápida y flexible, se puede estudiar mayor volumen de éstos y en menos tiempo mejorando así el diagnóstico final.

En el caso de los distribuidores de prensa, les interesa conocer el número de periódicos que han de llegar a una red de distribución diariamente, si se cuenta con métodos de modelación adecuados que se ajustan a este esquema y aprovechan su estructura, y a la vez son capaces de manejar adecuadamente su magnitud, se pueden desarrollar estimaciones y previsiones que dan como resultado la distribución óptima.

El valor del trabajo presentado radica en la incorporación al lenguaje de programación TOL de un nuevo conjunto de funciones y herramientas, que permiten al especialista analizar una mayor cantidad de datos en un menor tiempo y construir mejores modelos estadísticos que se ajustan más a la realidad, lo que le permite afrontar nuevas tareas y obtener mejores resultados en su labor. Además el lenguaje se enriquece con la incorporación de un conjunto de nuevas funciones y se obtienen un conjunto de bibliotecas y herramientas que se utilizan en otras aplicaciones.

Para explicar el desarrollo de cada uno de estos objetivos se estructuró este documento escrito en 3 capítulos.

En el capítulo 1, “Introducción al lenguaje de programación TOL” se explica las características de TOL, en especial lo referente a la manipulación de información dinámica y masiva; y se hace un recorrido por las facilidades que ofrece el lenguaje para la modelación estadística, concentrándonos principalmente en la etapa de estimación.

En el capítulo 2, “Análisis, diseño e implementación de las funcionalidades a incorporar” se analiza la propuesta de compactación para las tablas de datos, se revisan los aspectos referidos a la implementación del compactador, se explican las innovaciones que se introducen en la modelación con TOL y se describe la estructura computacional y algoritmos adoptados para representar y manipular un modelo jerárquico lineal.

En el capítulo 3, “Discusión de los resultados” se hace una descripción de los factores de compresión obtenidos por el compactador y se describen las funciones y herramientas nuevas incorporadas a TOL.

Finalmente se detallan las conclusiones del trabajo y se recomiendan algunos aspectos que como trabajo futuro deberían tenerse en cuenta para mejorar los resultados obtenidos.

Capítulo 1

"Introducción al lenguaje de programación TOL"

El lenguaje de programación TOL está orientado a la explotación de la información temporal, es especialmente adecuado cuando la información es muy dinámica y masiva. TOL proporciona una representación del tiempo, de las series temporales, de los modelos de comportamiento temporal, así como de otros tipos de objetos matemáticos necesarios para modelar y resolver problemas reales en el ámbito de la información dinámica.

El lenguaje ha ido evolucionado hasta nuestros días, pasando a ser en la actualidad parte de la comunidad de Software Libre, distribuyéndose bajo la licencia GNU GPL de la Free Software Foundation [http://www.gnu.org/] y se le ha dotado de unos recursos en Internet que le permiten contar con una comunidad de usuarios.

1.1 Elementos básicos del lenguaje.

Originariamente el lenguaje de implementación de TOL fue C y Fortran, pero terminó por adoptarse el lenguaje C++, y en la actualidad la implementación es una mezcla de C++ con la sintaxis tradicional de C.

Al adoptar la GPL, se tiene la gran ventaja de que se pueden usar otros productos también bajo GPL para fortalecer el lenguaje, así además de utilizar la biblioteca estándar de C++, la STL (Standard Template Library), se utiliza la GSL (GNU Scientific Library). Esta biblioteca de código ofrece algoritmos matemáticos para la resolución de problemas que TOL trataba de solucionar. Al adoptar su uso, se amplía la solidez de TOL con la potencia de un producto de gran éxito en la comunidad científico-matemática.

A principios de 2004 se puso en marcha TOL-Project.org, [http://www.tol-project.org/] un conjunto de recursos organizados alrededor de una Web que cubre las necesidades de la comunidad TOL: una Web principal, dos Webs de noticias (español e inglés), listas de correo, y un sistema para la gestión de informes de error, con esto se dota a TOL de una infraestructura que permite el libre acceso y distribución del mismo, de esta forma también las incidencias son comunicadas al equipo de desarrollo, y a su vez este puede comunicar las soluciones aplicadas, así como anunciar las nuevas prestaciones implementadas con las nuevas versiones.

El entorno de desarrollo de TOL, denominado TOLBASE, está disponible en WINDOWS y está programado en Tcl/Tk. También existen implementaciones para Sistemas

Operativos tipo Unix, como Linux, BSD y Mac OSX, se puede descargar TOL de forma libre y gratuita en formato fuente para todos los Sistemas Operativos, y en formato binario para Windows y Debian GNU/Linux.

TOL es un lenguaje declarativo, procedimental, interpretado, basado en dos puntos claves: [RUS05]

- especificación gramatical ligera, es decir, reglas sintácticas sencillas [AHO72].
- un potente conjunto de tipos de datos extensibles.

1.1.1 Tipos de datos.

TOL es un lenguaje orientado al manejo de series y conjuntos temporales, los tipos de datos que representan estas estructuras son *Serie* y *TimeSet* respectivamente, existen además otros tipos de datos que completan el lenguaje permitiendo realizar con él tareas de propósito general. En TOL todas las expresiones tienen un tipo de dato asociado, sin embargo algunas expresiones pueden obviar el tipo tomando como referencia el último que se ha empleado.

A continuación se enumeran y se explican brevemente los tipos de datos disponibles en el lenguaje TOL.

Textos (Text)

Una variable de tipo *Text* puede contener cualquier cadena de caracteres ASCII. Toda cadena entre comillas es un *Text*. TOL proporciona funciones para realizar búsqueda de cadenas y transformaciones en las variables *Text*. Ejemplo:

```
Text saludo = "Hola Mundo";
```

Números reales (*Real*)

Las variables de tipo *Real* son variables numéricas que se definen como reales de doble precisión. El tipo *Real* posee también el valor desconocido y el valor infinito. *Real* incorpora funciones de manipulación y transformación de reales: suma, producto, transformaciones trigonométricas, estadísticas, etc. Ejemplo:

```
Real fraction = 1/3;
```

Fechas (*Date*)

Es el tipo de datos que representa los instantes de tiempo. Se utiliza con frecuencia para establecer límites en las series temporales. El formato para crear variables de tipo *Date* es *yAAAAmMMdDD*. Ejemplo:

```
Date varDate = y2007m11d09;     // 9 de Noviembre del 2007
```

Números complejos (*Complex*)

Complex es el tipo de datos que representa los números complejos, ofreciendo un conjunto de operaciones y funciones que completan el álgebra de los números complejos. Ejemplo:

```
Complex Aurea = Sqrt(5) * i;
```

Matrices de números reales (*Matrix*)

Es el tipo de datos para la representación de matrices de números reales. Posee un completo conjunto de funciones de manipulación y transformación de matrices: suma, producto, transformación de Cholesky, resolución de sistemas lineales, etc. Ejemplo:

```
Matrix mat1 = ((0,0,1),(0,1,0),(1,0,0));
```

Conjuntos (*Set*)

El tipo de datos *Set* representa los conjuntos, o colecciones de elementos de cualquier tipo, incluyendo otros conjuntos, es el tipo de datos utilizado para el agrupamiento de variables, combinando conjuntos puede construir estructuras como vectores, tablas, árboles, etc.

El tipo de dato *Set* ofrece funciones '*SetOfTipo'* para crear conjuntos de cada tipo de dato. Existen las funciones *SetOfDate*, *SetOfMatrix*, *SetOfReal*, *SetOfSet*, etc. Ejemplo:

```
Set Data = SetOfSet(SetOfText("Hello","World"),SetOfReal(1,2,3));
```

La existencia de *Set* permite que los programas puedan agrupar las variables relacionadas entre sí por los criterios establecidos por el programador, evitando un panorama donde el número de variables y su identificación podrían llegar a hacer los grandes proyectos inmanejables.

Conjuntos estructurados

Una estructura es la definición de la composición de un conjunto, la estructura define el tipo de los elementos de un conjunto y asigna a cada uno de ellos un identificador, las estructuras facilitan la manipulación de grandes conjuntos de datos, al permitir acceder a los elementos de cada conjunto por el nombre del identificador de cada elemento.

Una estructura se define siguiendo el siguiente patrón:

Struct *struct-name* {

field-type id-campo [,

field-type id-campo]

};

struct-name: especifica el nombre de la estructura a crear.

field-type: tipo de dato asociado al elemento de la estructura.

id-campo: es el identificador del elemento en la estructura.

Ejemplo:

```
Struct stDBMySQL {
 Text driver,
 Text database,
 Text host
};
```

Conjuntos temporales (*TimeSet*)

El lenguaje TOL está basado en una representación real del tiempo social, el hecho de contar con una representación del tiempo tiene importantes implicaciones, ya que no sólo es posible manipular en TOL series temporales con periodicidades habituales diarias, semanales, mensuales, anuales, etc., sino que también es posible la descripción de periodicidades irregulares, por ejemplo, cierto o ciertos días de la semana, ciertos meses o ciertos días del mes, combinaciones de días concretos de la semana por meses del año, días concretos del año tanto sueltos como por intervalos, períodos festivos, etc. que pueden definirse de forma adecuada para cada problema en concreto.

Se debe notar que estas periodicidades irregulares pueden ser la forma natural de ver los datos y la información histórica en muchas organizaciones, la realidad económica y social es muy rica en ejemplos de este tipo. La representación del tiempo diseñada en TOL pretende recoger toda esta casuística que de forma natural se presenta en el mundo real, especialmente en el mundo económico y social, de forma que cada aplicación concreta pueda contar con su propia definición de la periodicidad de sus datos.

El tipo de datos *TimeSet* ofrece un completo conjunto de operaciones que permiten construir nuevos conjuntos temporales a partir de otros ya creados, *TimeSet* permite hacer operaciones de unión, intersección y diferencia de conjuntos temporales. Ejemplo:

```
TimeSet losLunesdeFebrero = WD(1) * M(2);
```

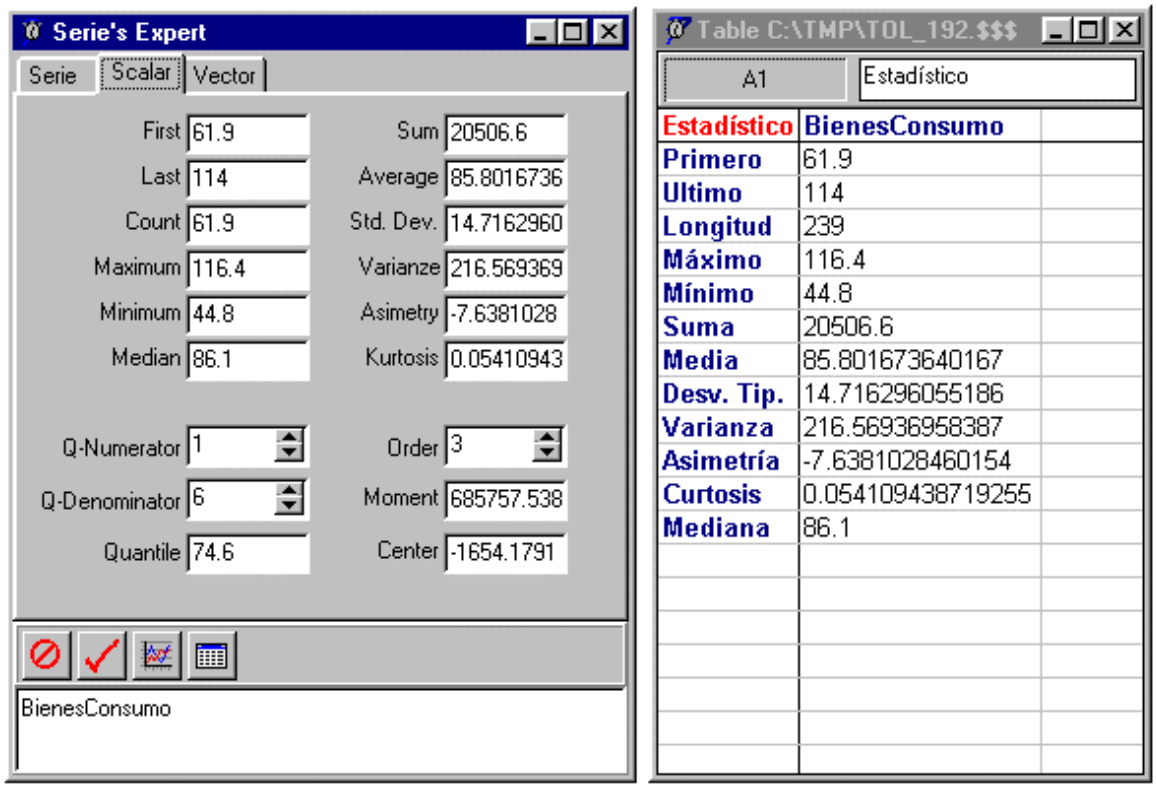
El tiempo social (años, meses, semanas, días, etc.) tiene una influencia fundamental sobre toda la actividad humana y muy especialmente sobre la actividad económica, por ejemplo, los períodos de vacaciones, los desplazamientos estacionales de la población, las ofertas de fin de temporada, los cierres de año, etc. Por otra parte, el aumento de la competencia hace que cada vez sea más necesario un seguimiento puntual de los diferentes procesos económicos, industriales, de evolución del mercado y sus tendencias, etc. Este hecho tiene su reflejo en el aumento de popularidad de técnicas como el “just in time” o el “time to market”.

Series temporales (*Serie*)

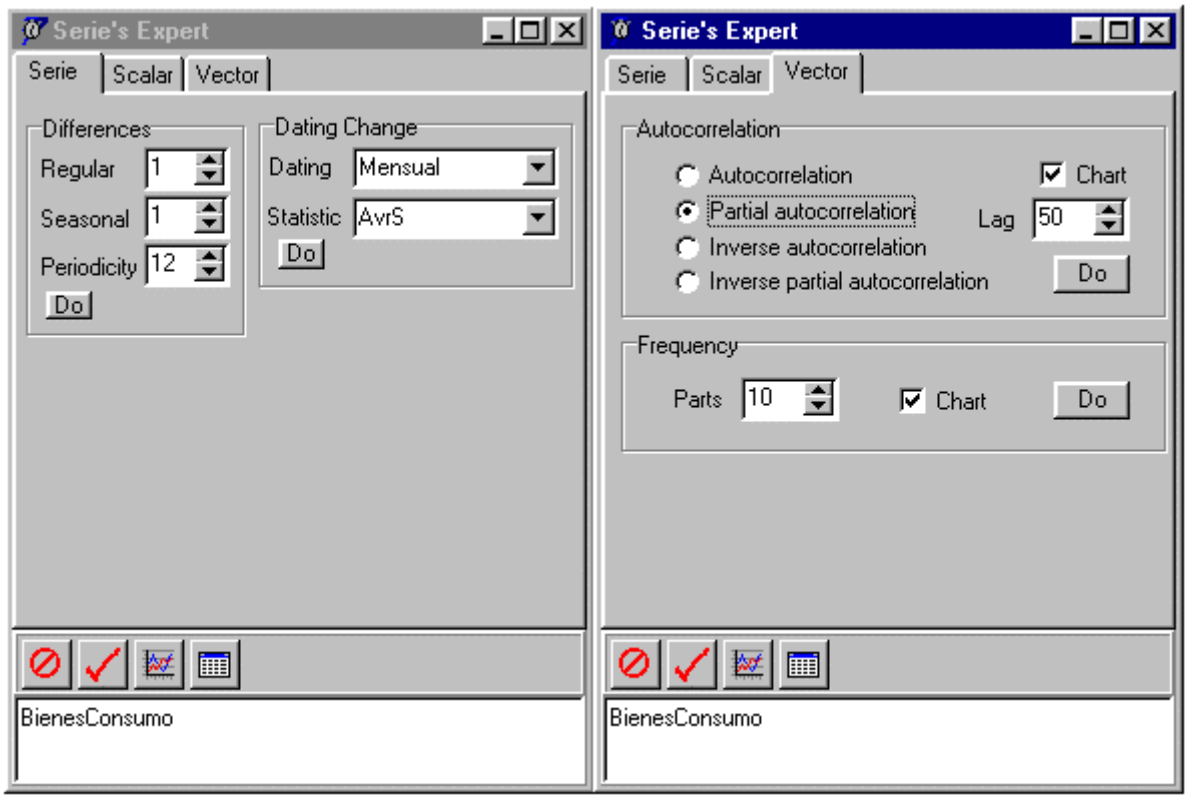
El tipo de datos *Serie* permite la creación y manipulación algebraica de series temporales para realizar operaciones, gráficos, estimaciones y previsión de datos. Ejemplo:

```
Set sData = Include("series.bdt"); // Inclusión de series S1 y S2
Serie bas01 = S1+S2;
```

La interfaz de usuario gráfica TOLBASE está dotada de una potente capacidad para generar gráficos lo que permite obtener el máximo provecho de TOL. También se pueden obtener algunos de los estadísticos más relevantes como funciones de series temporales en el espacio de los números reales:



La ventana del diálogo experto en series temporales posee varias solapas (subcarpetas) a través de las cuales se pueden realizar diferentes acciones para la serie, por ejemplo, una de las acciones posibles es obtener gráficos de autocorrelación y autocorrelación parcial y/o inversa sobre la serie temporal seleccionada. Esto será visto en más detalle en la sección de introducción a la modelación.



TOL posee también funciones para la creación de series deterministas básicas, también denominadas series artificiales, como el pulso, la compensación, el escalón y la tendencia. Estas series se emplean en el análisis de intervención para captar y filtrar comportamientos anómalos, hechos esporádicos o simplemente efectos calendarios en las series observadas.

Polinomios (*Polyn*)

El tipo *Polyn* permite la representación de polinomios en TOL, la principal utilidad es que permiten el desplazamiento temporal de las series a través de polinomios con el operador B de retardo (backward) o con operador F de adelanto (forward). Ejemplo:

```
Polyn stepwise = F+B^2+F^3+B^4+F^5+B^6+F^7;
```

Fracciones de polinomios (*Ration*)

El tipo *Ration* representa las fracciones racionales de polinomios. Una función racional de retardos se define como un cociente de polinomios de retardo. Su principal utilidad es resolver ecuaciones en diferencias. TOL permite operaciones entre fracciones racionales tales como el producto, suma, potencias, etc. Ejemplo:

```
Ratio fraction = F/(1-B^5);
```

Código TOL (*Code*)

Es el tipo de dato asociado a las funciones "built-in", es decir, las funciones proporcionadas por el lenguaje, y a las funciones creadas por el usuario, puede ser utilizado para declarar objetos como parámetro de entrada o de retorno de las funciones, esta propiedad permite a TOL manipularse a sí mismo. Ejemplo:

```
Real sumar(Real a, Real b) { a+b };
Code miFunc = sumar;
miFunc(3,1);
```

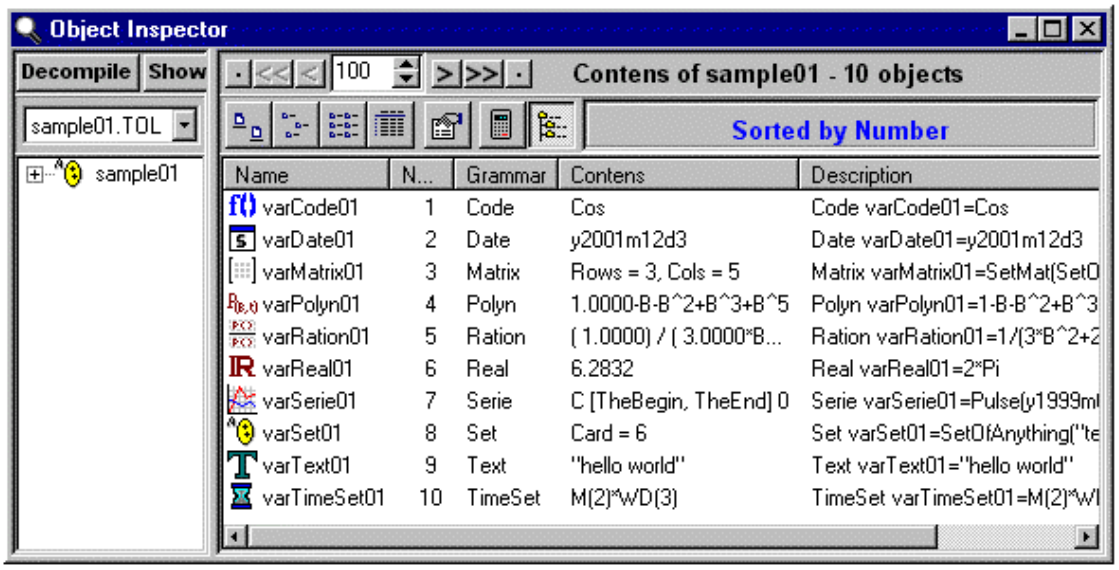
Anything

Es el tipo de dato de las funciones especiales:

Case, Do, Eval, Field, Find, If, While, etc.

A través del inspector de objetos del entorno TOLBASE se pueden consultar los diferentes objetos creados como resultado de la ejecución de un fichero TOL, para cada objeto se muestran sus propiedades, cada fichero TOL se representa como un conjunto que contiene todos los objetos declarados dentro de él.

En la siguiente figura se muestran los objetos declarados dentro del fichero *sample01.tol* siendo *sample01* el nombre del conjunto asociado al fichero:



1.1.2 Funciones de entrada/salida.

TOL posee sus propios métodos de almacenamiento y recuperación de la información, así cuenta con una interfaz para leer y escribir datos desde ficheros, las funciones de lectura crean un objeto de tipo *Set* el cual contiene los datos leídos, ya sean éstos series temporales, matrices, tablas o cualquier otro código.

Para incluir ficheros de datos se utiliza la función *Include,* la cual está especializada en dependencia del tipo de fichero que puede leer como se verá a continuación.

```
Set Include(Text Filename);
```

Usado para incluir código TOL, el resultado de la evaluación del código contenido en el fichero *Filename* forma parte del conjunto que retorna la sentencia *Include*. Esta es la manera óptima de cargar funciones que se quieran tener disponibles para determinados programas.

```
Set IncludeBDT(Text Filename);
```

Incluye el conjunto de series temporales con el mismo fechado y entre las mismas fechas definidas en el fichero *Filename* con formato BDT (Bayes Data Table).

```
Set IncludeBMT(Text Filename);
```

Devuelve un conjunto cuyo único elemento es la matriz de números reales guardada en un fichero con formato BMT (Bayes Matrix Table).

```
Set IncludeBST(Text Filename);
```

Es la función utilizada para el almacenamiento de estructuras de datos de cualquier tipo en forma de tabla, devuelve el conjunto de todos los conjuntos estructurados de un fichero con formato BST. Ejemplo:

Partiendo de una estructura en TOL como la siguiente:

```
Struct str_datPers {
 Text f_nombre,
 Date f_nacimiento,
 Text f_procedencia,
 Real f_estatura,
 Real f_peso
};
```

Y un fichero llamado *dataset.bst* con el mismo formato y contenido que se muestra a continuación:

nombre;	nacimiento;	procedencia;	estatura;	peso;
"Jorge";	y1963m11d01;	"Venecia";	1.77;	73.0;
"Pablo";	y1974m05d23;	"Roma";	1.83;	72.0;
"Juan";	y1981m01d02;	"Madrid";	1.78;	74.0;

La siguiente sentencia:

```
Set fileBST = IncludeBST("/home/youruser/TOL/dataset.bst");
```

Da como resultado:

```
[[ [[ "Jorge", y1963m11d01, "Venecia", 1.770000, 73.000000 ]],
   [[ "Pablo", y1974m05d23, "Roma", 1.830000, 72.000000 ]],
   [[ "Juan", y1981m01d02, "Madrid", 1.780000, 74.000000 ]] ]]
```

El conjunto *fileBST* contiene 3 elementos, cada uno de ellos es un conjunto con 5 elementos, nombre, fecha de nacimiento, lugar de procedencia, estatura y peso, recogidos en cada línea del fichero BST.

1.2 La compactación en TOL.

Los datos a que se hacía referencia anteriormente, cuando se hablaba de las compañías telefónicas, vienen expresados de esta manera, así que una de las tareas consistió precisamente en mejorar la forma habitual en que se leían los datos con esta función, lo cual era insuficiente para los propósitos de análisis de los mismos.

La manipulación y consulta de este gran volumen de información debía replantearse de manera tal que fuera posible tener disponible la mayor cantidad de ficheros con esta información, por lo que lograr un gran factor de compresión es un aspecto importante, pero también es necesario no descuidar la velocidad con que este fichero una vez compactado debe descompactarse para poder efectuar las consultas necesarias.

Antes de exponer la solución dada a este problema, se debe mencionar que anteriormente ya se había hecho por los desarrolladores de TOL, un intento para resolver esta problemática denominado HBF (Huffman by Fields) [BUE02], el mismo consiste en compactar las columnas de la tabla una por una utilizando el algoritmo de Huffman, de ahí su nombre, más adelante se hace referencia a este algoritmo.

Está claro que un sistema de codificación como este es estrictamente secuencial, por tanto la decodificación es también secuencial, lo que puede suponer falta de eficiencia cuando las secuencias son muy largas.

A pesar de haber logrado factores de compresión adecuados, aunque no exactamente los esperados, esta variante presenta la dificultad de que se necesita descompactar el fichero completo para poder efectuar consultas en algún registro, pues como es lógico la información es compactada por columnas, esto sugirió la idea de que el nuevo método debe aprovechar la estructura por filas de la tabla, es decir la interrelación y/o dependencia entre los campos y no la simple aplicación de Huffman por campos independientes como lo hace HBF.

También de esta experiencia se desprende que el nuevo método debe lograr acceder a partes específicas del fichero compactado que se corresponden con los registros de la

tabla que se necesitan consultar, es decir lograr un acceso aleatorio y no secuencial como el logrado en HBF, evitando así la necesidad de descompactar el fichero completo y logrando por tanto mayor velocidad en el análisis de los datos.

Por último la propuesta debe ser consecuente con la filosofía de TOL respecto a su orientación al tiempo, esto asegura que los resultados no dejarán de ser utilizados prematuramente.

Se Introducen primeramente aquí algunos conceptos importantes de la teoría de la compactación, luego en el capítulo 2, se explica la estructura de las tablas de las compañías telefónicas que se necesitan compactar y, considerando todos los aspectos mencionados anteriormente, se expone la propuesta que se hace y su implementación.

La teoría de compactación de la información [SAY00], [SAL00] parte de que la información está representada por una sucesión, llamada mensaje, de unidades llamadas letras, el conjunto de todas las posibles letras en un mensaje se llama alfabeto, la longitud de un mensaje está dada por la memoria física digital que ocupan, es decir la cantidad de bits (unidad binaria de información). La teoría de compactación propone reescribir los mensajes utilizando un nuevo alfabeto con el cual la longitud del mensaje sea menor a la longitud del mensaje original.

Se llama diccionario o código a una función biyectiva que hace corresponder a cada letra una representación o cadena de bits, llamada letra de código, que la identifique unívocamente, la sustitución de las letras originales por las letras de código debe ser un proceso reversible, esto es garantizado por los códigos de prefijo. El algoritmo de Huffman que se utiliza es un algoritmo de prefijo, lo cual quiere decir que dos letras del código no comparten primeros bits iguales.

El algoritmo de Huffman [MOF97] se utiliza para la codificación de datos en los cuales no se asume ninguna estructura particular, es decir, se asume independencia o no correlación entre las apariciones de las letras del alfabeto en la sucesión a compactar, solamente se asumen que estas aparecen con determinada frecuencia. El algoritmo de Huffman está basado en dos observaciones fundamentales:

- En un algoritmo óptimo las letras que ocurren con más frecuencia tendrán palabras de código más cortas.
- En un código óptimo las dos letras de menos frecuencia son codificadas por palabras de código de igual longitud.

A partir de las frecuencias de aparición se le hace corresponder a cada letra una secuencia de bits, 0 y 1's respetando ser códigos de prefijo, este diccionario se almacena físicamente en un árbol completo binario.

1.3 La modelación con TOL.

En este apartado se hace una breve introducción a la modelación con TOL, y cómo emplear los modelos que mejor se ajusten para la previsión de datos. En un proceso de modelación se deben seguir los siguientes pasos:

- Análisis y planteamiento del problema
- Lectura de los datos
- Identificación del modelo
- Estimación
- Validación
- Previsión

El próximo punto que se aborda en este trabajo, cae precisamente en una de estas etapas, la estimación del modelo, motivado como se ha dicho antes por la resolución de los modelos de las distribuidoras de prensa, es por ello que parece oportuno hacer un breve recorrido por la modelación con TOL.

La mayor parte de las series temporales están generadas por modelos no estacionarios siendo necesarias transformaciones que las conviertan en procesos estacionarios, con el objetivo de utilizar las ventajas que ofrecen estos últimos para su modelación [CHE00], [PEÑ89].

En una primera fase de la elaboración de un modelo ARIMA [BOX94] se requieren fundamentalmente como instrumentos básicos de identificación la función de autocorrelación simple y parcial estimadas.

Una vez identificado el modelo se obtienen unos valores estimados para los parámetros, y se lleva a cabo la fase de validación que va dirigida a establecer si se produce o no esa adecuación entre datos y modelo.

Finalmente, en la fase de predicción, se realizan pronósticos en términos probabilísticos de valores futuros de la variable.

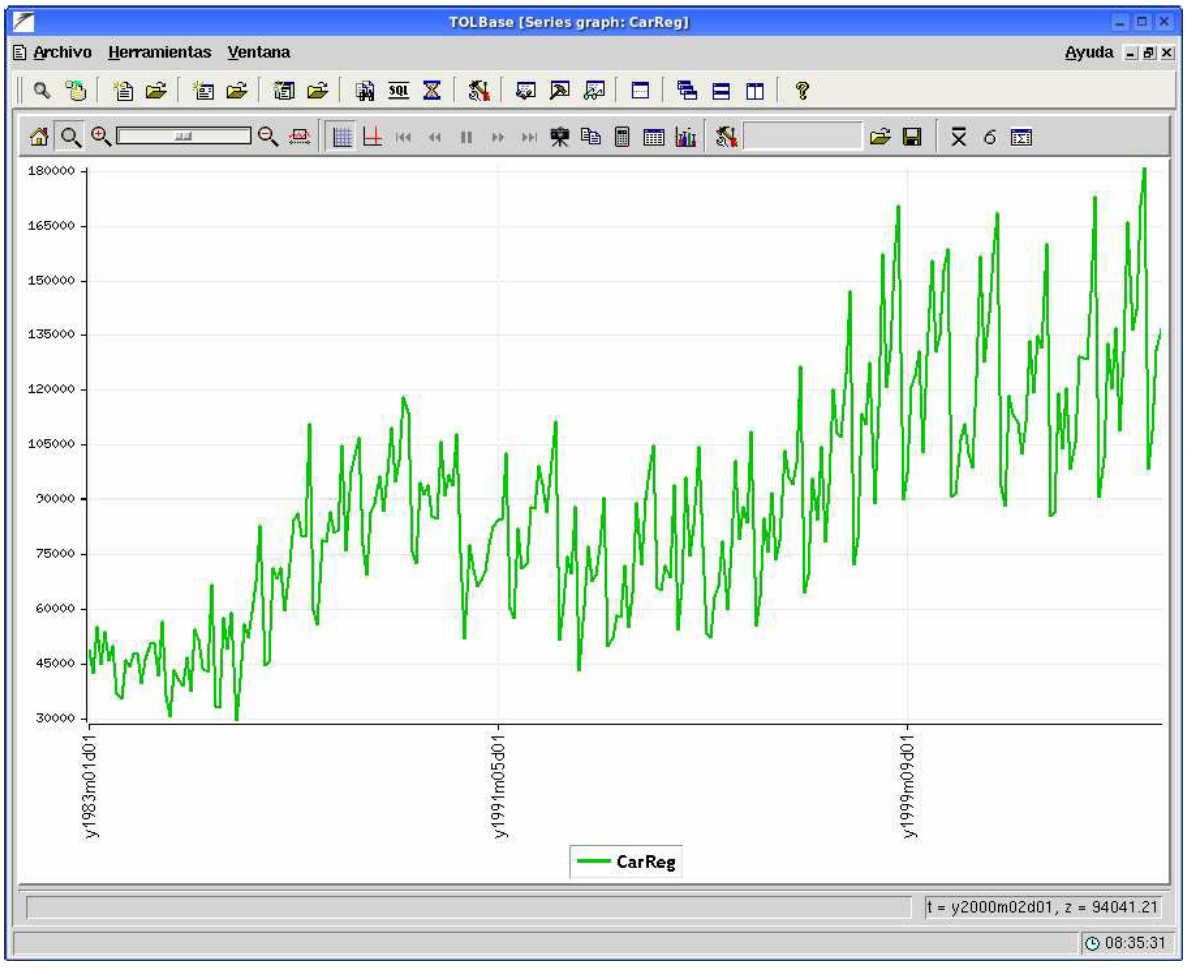
A continuación a modo de ejemplo se hace un resumen de la modelación de la evolución de la matriculación de coches durante el período Enero de 1983 a Noviembre de 2004 en España.

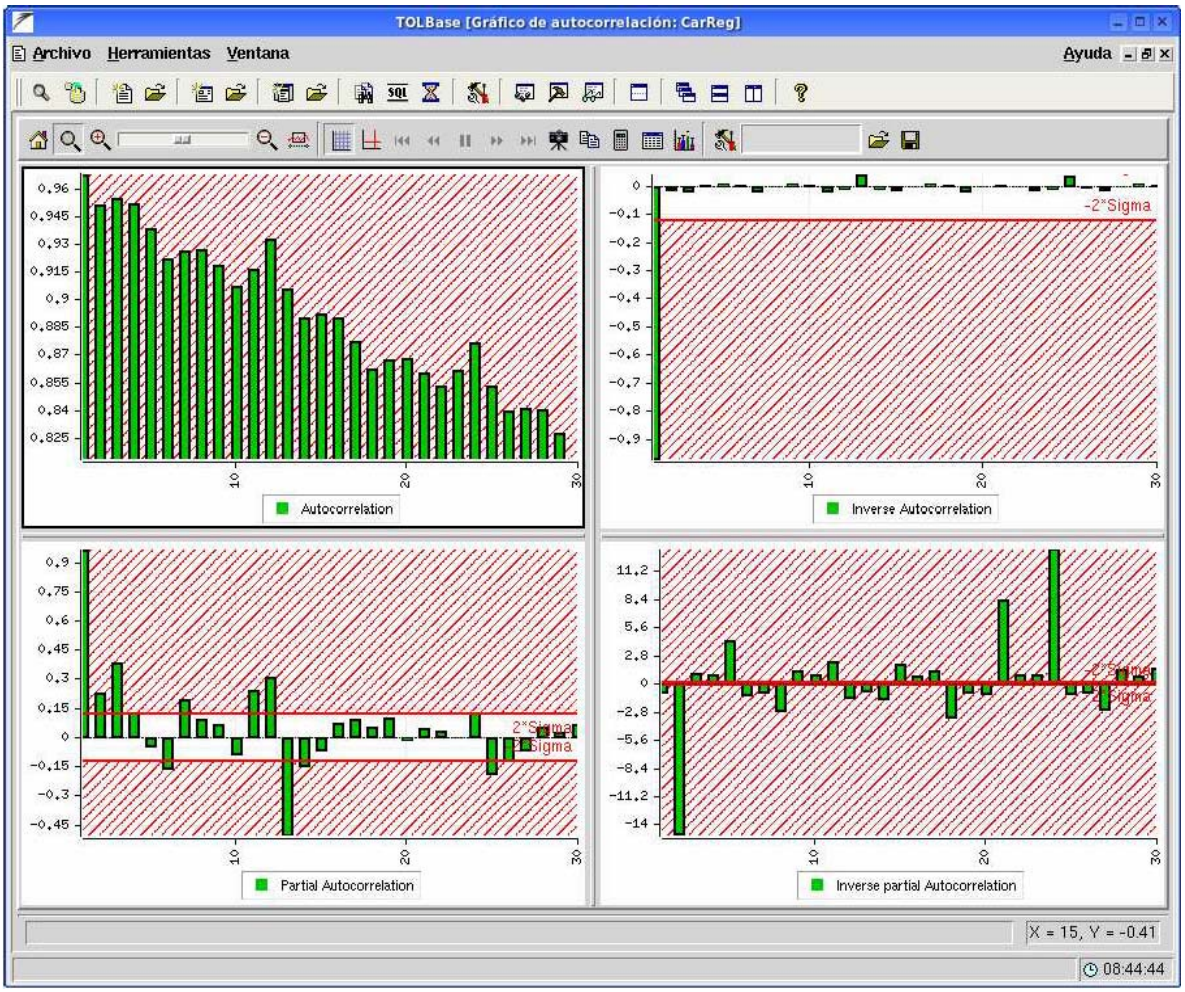
La modelación se ha llevado a cabo con TOLBASE, la interfaz gráfica de TOL, dotada de la capacidad de graficar series temporales y es la interfaz gráfica por excelencia para la programación en TOL [RUS95].

La lectura de datos de una serie puede hacerse bien a través de la interfaz con bases de datos, o bien a través de ficheros, la lectura a través de ficheros puede realizarse con la función *IncludeBDT* como se explicó en el apartado de la introducción al lenguaje TOL.

1.3.1 Fase de identificación (visualización).

Una vez incluido el fichero y haciendo uso de las facilidades de TOLBASE, se observa el gráfico de la serie CarReg y los gráficos de autocorrelación simple y parcial de la serie.



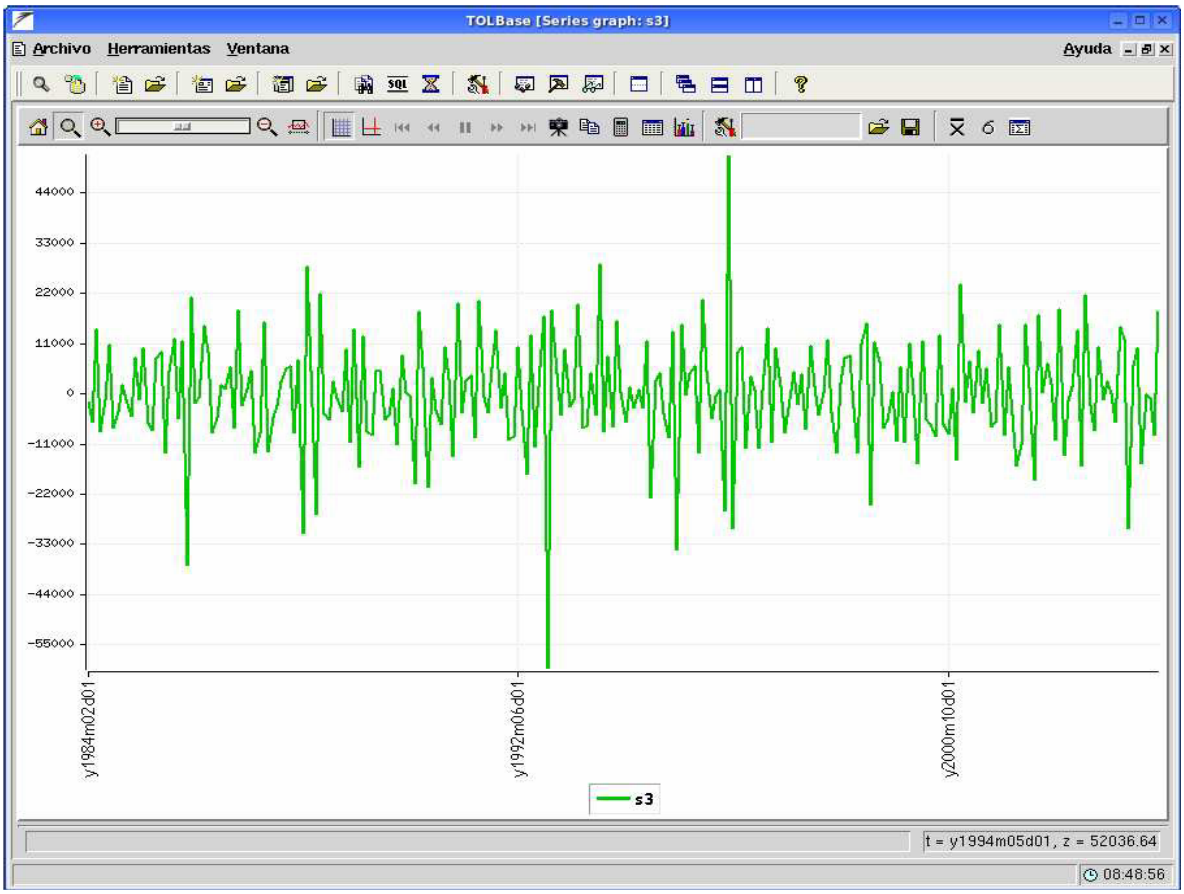


A la vista de los gráficos de las funciones de autocorrelación simple (FAS) y parcial (FAP) de la serie, se decide aplicar una diferencia regular de orden 1, una diferencia estacional anual y establecer el modelo que parece más apropiado.

Para hacer este tipo de transformaciones se tienen que aplicar ciertos tipos de polinomios a las series.

```
Polyn diff = (1-B) * (1-B^12);
Serie s3 = diff:CarReg;
```

En la figura se muestra la serie *s3* libre de la estacionalidad y tendencia.



Se trata de un proceso estacionario donde se observan una serie de datos atípicos que se necesitan intervenir a través de variables artificiales.

Se definen entonces las siguientes variables (series deterministas) que se introducen en el modelo y cuyo objetivo es captar todos estos efectos:

```
Serie pulse9607 = Pulse(y1996m07, Monthly);
Serie pulse9301 = Pulse(y1993m01, Monthly);
Serie pulse8601 = Pulse(y1986m01, Monthly);
```

En la serie diferenciada estacional y regularmente se observa de nuevo su FAS y FAP para decidir los posibles modelos a estimar y a la vista de estos gráficos se propone ajustar la serie a un proceso SARIMA (2, 1, 0) x $(0, 1, 1)_{12}$

1.3.2 Fase de estimación.

Una vez identificado el modelo que sigue a la serie, se procede a estimar los parámetros, para ello, TOL dispone de la función:

```
Set Estimate(Set modelDef [,Date desde, Date hasta],
             Set parametrosAPriori);
```

Esta función requiere, como primer argumento, un conjunto como el que se describe a continuación:

```
Set ModelDef(Serie Output,
             Real  FstTransfor,
             Real  SndTransfor,
             Real  Period,
             Real  Constant,
             Polyn Dif,
             Set   AR,
             Set   MA,
             Set   Input,
             Set   NonLinInput)
```

donde los parámetros representan:

Output: es la serie que se pretende ajustar.

FstTransfor: es un número real que representa la primera transformación.

SndTransfor: es un número real que representa la segunda transformación.

Period: representa la estacionalidad que presenta la serie.

Constant: si se requiere ajustar el modelo con una constante.

Dif: es el producto de los polinomios diferencia regular y diferencia estacional que se necesita aplicar a la serie para convertirla en una serie estacionaria.

AR: es un conjunto de polinomios con dos elementos, el primero es el polinomio AR de la parte regular y el segundo es el polinomio AR de la parte estacional.

MA: es un conjunto de polinomios con dos elementos, el primero es el polinomio MA de la parte regular y el segundo es el polinomio MA de la parte estacional.

Input: es el conjunto de variables explicativas (por ejemplo intervenciones).

InputNonLin: representa el conjunto de inputs no lineales.

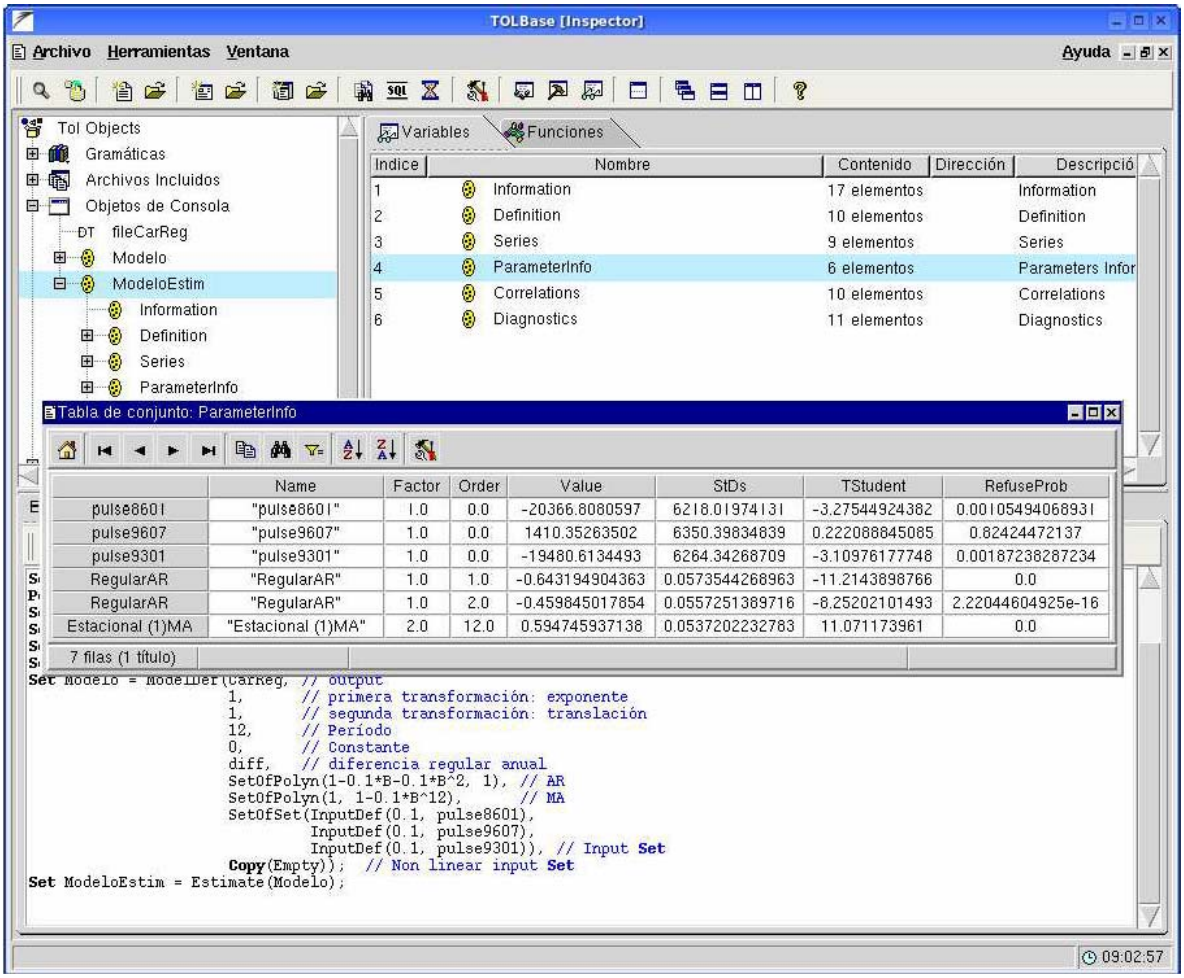
que en el caso actual es:

```
Set M = ModelDef(CarReg, // output
                 1,      // primera transformación: exponente
                 1,      // segunda transformación: traslación
                 12,     // Período
                 0,      // Constante
                 diff,   // diferencia regular anual
                 SetOfPolyn(1-0.1*B-0.1*B^2, 1),   // AR
```

```
        SetOfPolyn(1, 1-0.1*B^12),        // MA
        SetOfSet(InputDef(0.1, pulse8601),
                 InputDef(0.1, pulse9607),
                 InputDef(0.1, pulse9301)), // Input Set
        Copy(Empty)  // Non linear input Set
      );
```

Como segundo y tercer argumento (opcionales) las fechas inicial y final que se quiere tener en cuenta para realizar la estimación, en general se utilizan todos los datos disponibles, como último argumento, también opcional, se utiliza *parametrosAPriori*.

Una vez compilado se obtiene la salida de la figura:



1.3.3 Fase de validación.

El objetivo perseguido al elaborar un modelo ARIMA es encontrar un modelo que sea lo más adecuado posible para representar el comportamiento de la serie estudiada, así un modelo ideal es el que cumpla los siguientes requisitos: [BOX94]

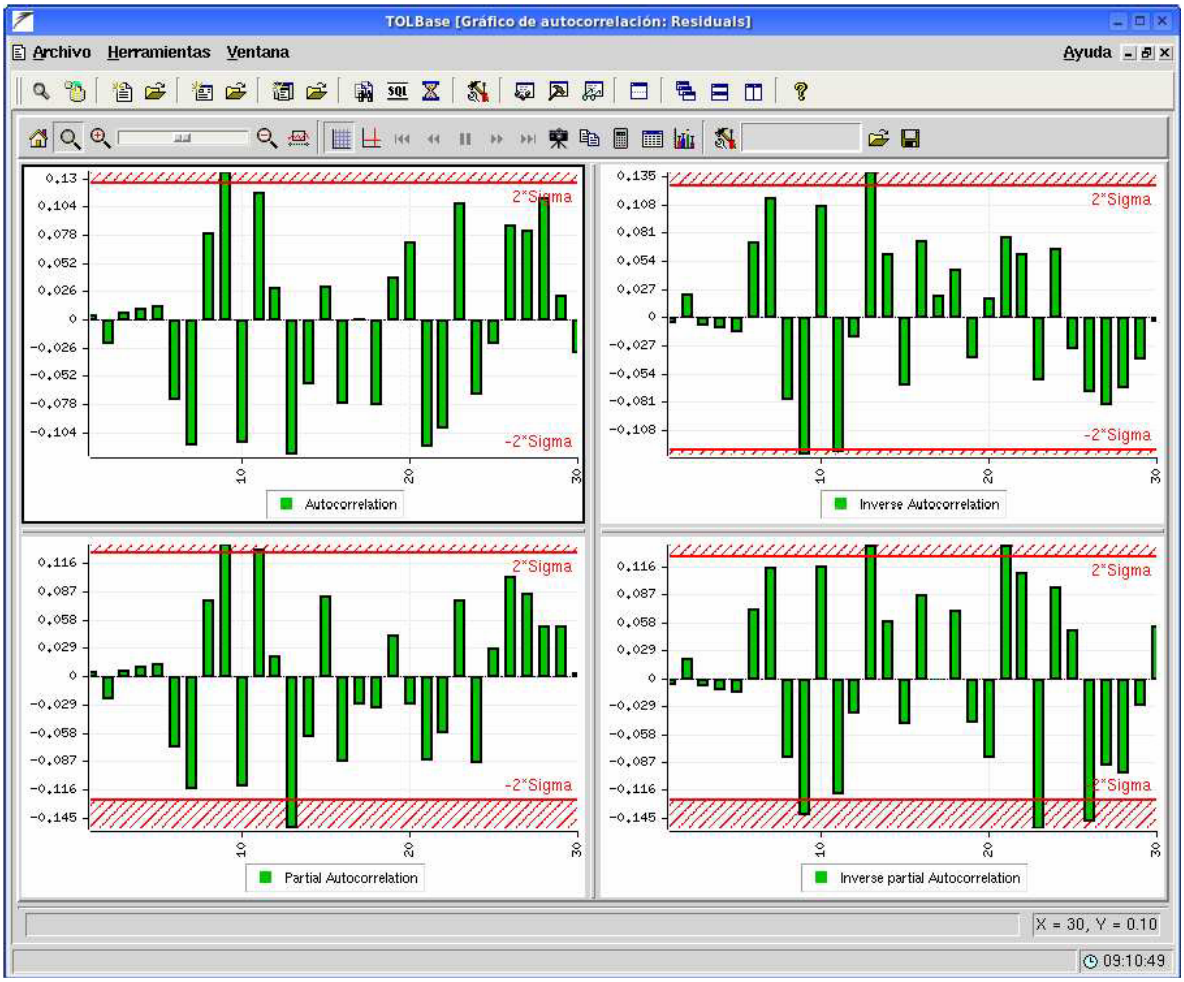
- Los residuos del modelo estimado se aproximan al comportamiento de un ruido blanco.
- El modelo estimado es estacionario e invertible.
- Los coeficientes son estadísticamente significativos, y están poco correlacionados entre sí.
- El grado de ajuste es elevado en comparación al de otros modelos alternativos.

La finalidad de la fase de validación, consiste precisamente en analizar la adecuación entre el modelo y los datos, para ello se realiza un análisis de los residuos que consiste en comprobar que los residuos se comportan como un ruido blanco.

Para la fase de validación se utiliza la serie de los residuos:

```
Serie ModeloEstim->Series->Residuals
```

Si se representa su FAS y FAP se observa que se corresponden a las de un ruido blanco:



1.3.4 Fase de predicción.

Las tres primeras fases de la elaboración de un modelo ARIMA constituyen un proceso iterativo cuyo resultado final es la obtención de un modelo estimado que sea compatible con la estructura de los datos. Una vez que se ha conseguido este resultado, la fase siguiente consiste en utilizar este modelo estimado en la predicción de valores futuros de la variable objeto de estudio.

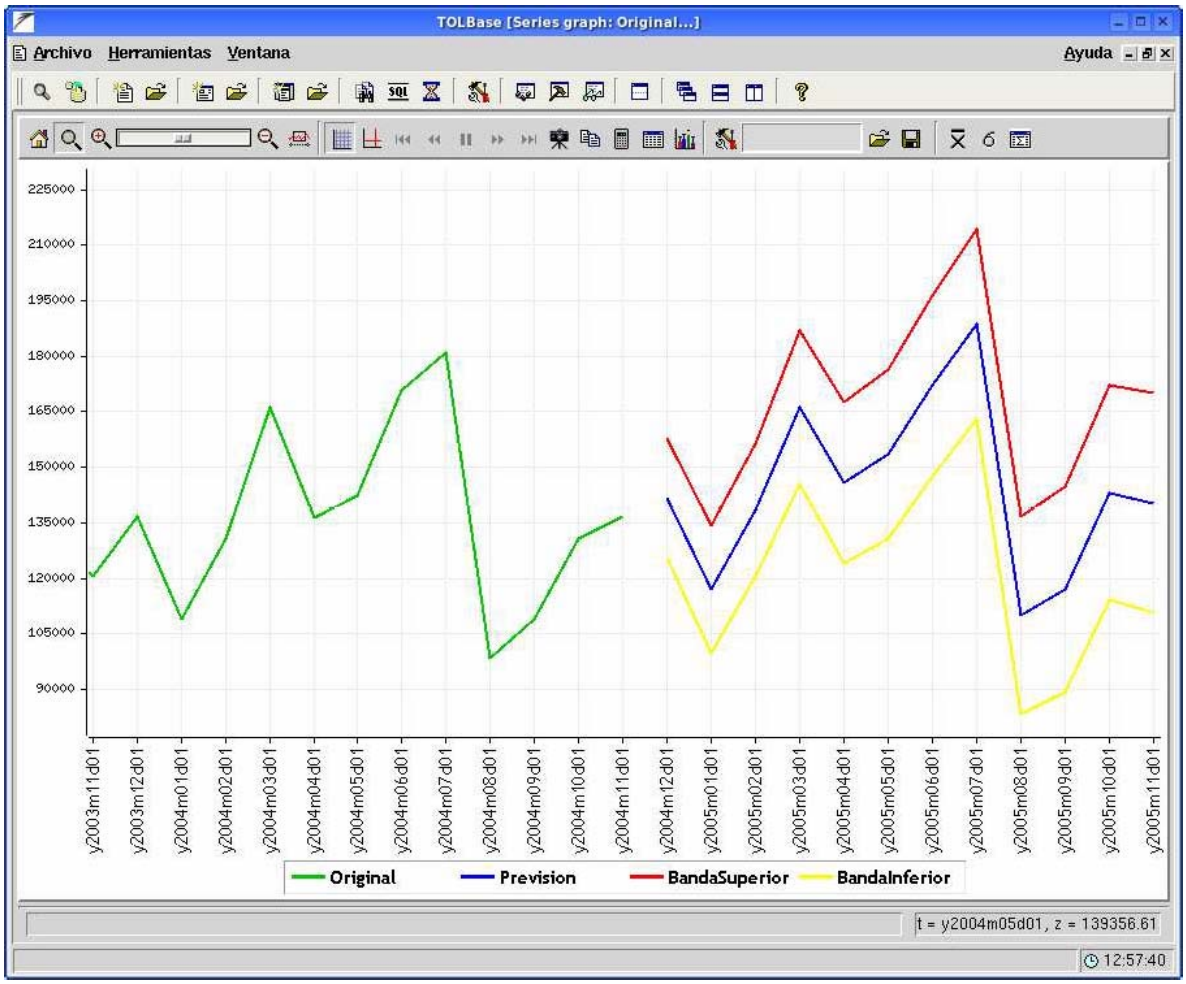
Por ejemplo, se quiere predecir el año siguiente al término de los datos de la serie. Para ello, se emplea la función:

```
CalcForecasting(Set modeloEstim, Date iniSerie, Date finSerie,
                Real numPrevisiones, Real alfa)
```

donde alfa es el ajuste de las bandas de confianza. En el ejemplo es:

```
Set ModeloPrev = CalcForecasting(ModeloEstim, First(CarReg),
                                 Last(CarReg), 12, 0.05);
```

El gráfico de la figura muestra la previsión del próximo año con los datos de los dos últimos años y las bandas de confianza al 95%:



Como se ve en la fase de estimación, TOL dispone de la función *Estimate* (Estimador máximo-verosímil de modelos ARIMA), para estimar los parámetros del modelo que sigue la serie que se pretende ajustar, pero esto es sólo para una serie, en el problema de la distribución de prensa, una serie reflejaría cómo se comporta la distribución en un solo punto de venta (kiosco), sin embrago se trata aquí de que muchos de estas series comparten parámetros comunes que explican su comportamiento o incluso más, se necesita hallar las relaciones existentes entre parámetros de diferentes kioscos.

1.4 Los grafos en la construcción de modelos jerárquicos.

En esta sección se establecen conceptos de la teoría de grafos que son necesarios para el desarrollo del resto del documento, ya que la solución dada al problema de prensa, como se verá en el capítulo 2, utiliza esta estructura para representar la información a modelar. Una revisión rápida de estos conceptos puede verse también en [WIK07b] y si se quiere profundizar en los elementos de teoría de grafo es recomendable leer [CHA96].

Definición 1. Un grafo G dirigido es un par ordenado $G:=(N, E)$, donde:

- N es un conjunto cuyos elementos se denominan nodos o vértices.
- E es un conjunto de pares ordenados de nodos, denominados aristas dirigidas o arcos.

En un arco del grafo $e_{ij} = (N_i, N_j)$ se identifica:

- origen del arco $s(e_{ij}) = N_i$
- destino del arco $d(e_{ij}) = N_j$

Definición 2. Un ciclo en un grafo dirigido $G:=(N, E)$, es una secuencia de arcos $C = (e_1, \ldots, e_k)$ tal que:

- $s(e_1) = d(e_k)$
- $\forall\ 1 < j \leq k,\ d(e_{j-1}) = s(e_j)$

Definición 3. Un grafo DAG $G:=(N, E)$ es un grafo dirigido para el cual no existe un ciclo.

A partir de aquí se pueden definir algunos atributos asociados a los nodos y aristas de un DAG.

Definición 4. El conjunto de hijos de un nodo N_i es $ch(N_i) = \{ N_j \mid \exists\ e_{ij} = (N_i, N_j) \in E \}$.

Asociado al concepto de hijos de un nodo se tiene el grado de salida:

Definición 5. El grado de salida de un nodo N_i se define como $dout(N_i) = |ch(N_i)|$. Donde $|.|$ denota el cardinal del conjunto.

Similar al concepto de hijos de un nodo se tiene el de padres:

Definición 6. El conjunto de padres de un nodo N_i es $pa(N_i) = \{ N_j \mid \exists\ e_{ji} = (N_j, N_i) \in E \}$.

Asociado al concepto de padres de un nodo se tiene el grado de entrada:

Definición 7. El grado de entrada de un nodo N_i se define como $din(N_i) = |pa(N_i)|$. Donde |.| denota el cardinal del conjunto.

Definición 8. El conjunto de nodos hermanos de un nodo N_i se define como $sib(N_i) = \{ N_j \mid \exists N_k \in N \wedge \{N_i, N_j\} \subset ch(N_k) \}$

Capítulo 2
"Análisis, diseño e implementación de las funcionalidades a incorporar"

Como se había anunciado anteriormente, en este capítulo se describe la estructura de las tablas que se deben compactar, se explica la propuesta de compactación que se hizo, y finalmente se revisan los aspectos referidos a la implementación del compactador.

Más adelante se explican las innovaciones que se introducen en la modelación con TOL para resolver el problema de la distribución en prensa, se describe la estructura computacional adoptada para representar y manipular un modelo jerárquico lineal y se dan las descripciones algorítmicas necesarias en la estimación de estos modelos.

2.1 Descripción de la estructura de las tablas.

Los nombres de los campos y su significado son los siguientes:

Campo	Contenido
id_cliente_old	Código del cliente anterior o inexistente
id_cliente	Código del cliente
id_subsegmento_cliente	Subsegmento del cliente, preclasificación
id_localidade_origem	Identificador de localidad de origen de la llamada
nrc	Línea por la que se hace la llamada
numero_origem	Número llamante
id_localidade_destino	Identificador de localidad de destino de la llamada
numero_destino	Número llamado
id_tipo_chamada	Saliente o entrante
id_degrau_tarifario	Código de tarifa entre origen y destino
id_tipo_trafego	Interestatal, local, etc.
id_critica	Si la llamada se finalizó con éxito o no
id_carrier	Código de la prestadora del servicio
data_hora_chamada	Hora de la llamada
segundos_duracao_real	Duración de la llamada
id_data_carga	Fecha en que se carga la llamada

La siguiente tabla muestra los distintos campos según el tipo de dato utilizado por un gestor de datos en particular, la memoria física que ocupan, así como ejemplos de los valores que toman.

Campo	Tipo	Bytes	Bits	Ejemplos
id_cliente_old	int	4	32	0, 38454108

id_cliente	int	4	32	45255, 38454108
id_subsegmento_cliente	smallint	2	16	564, 632
id_localidade_origem	smallint	2	16	11664
nrc	bigint	8	64	4664688490
numero_origem	bigint	8	64	111353267496200
id_localidade_destino	smallint	2	16	11664
numero_destino	bigint	8	64	270383522259600
id_tipo_chamada	smallint	2	16	0,1,2
id_degrau_tarifario	smallint	2	16	0,1, …, 6
id_tipo_trafego	smallint	2	16	27,28
id_critica	tinyint	1	8	2
id_carrier	smallint	2	16	0,1, …,20
data_hora_chamada	smallint	2	16	-18826
segundos_duracao_real	int	4	32	2299
id_data_carga	smallint	2	16	-17984
Total		55	440	

2.2 Análisis de la propuesta de compactación.

Los registros de llamadas o vectores de información sobre el proceso de una llamada, pueden ser vistos en una primera aproximación como el proceso de la interacción entre la fuente que decide la llamada (el cliente), y el medio (condiciones de la llamada). Este proceso transcurre en el tiempo, y cuenta de factores estáticos y dinámicos. Captar el comportamiento del cliente para ser utilizado en la compactación pasa por el problema de encontrar el balance entre la cantidad de información que se quiere o puede utilizar del cliente y la velocidad de acceder a esta información.

En la siguiente sección se hace una clasificación de los distintos campos a partir de las características de los mismos, esta clasificación define la esencia de la propuesta en que los campos se compactarán según sus características probabilísticas, dicha clasificación está orientada a la futura creación de un protocolo que permita especificar por el usuario qué métodos aplicar para comprimir futuras bases de datos de tipos relativamente equivalentes.

Es importante notar aunque no se profundice en ello, que para hacer la propuesta se tuvieron en cuenta las características estadísticas que se pueden observar de los datos, basado en dichas características es que se adoptan los distintos esquemas de

modelación de la estructura de los datos y se sugieren los algoritmos de compactación [ART04].

2.2.1 Compactación de campos temporales.

Por campos temporales se entiende aquellos cuya estructura depende del factor tiempo y no de un cliente individual como fuente de decisión. En principio se parte de que las fuentes son no estacionarias con respecto al tiempo, evolucionando en el mismo, por esta razón el primer paso a seguir en el proceso de compactación es ordenar los registros temporalmente, es decir, basados en el campo *data_hora_chamada*, este campo es entonces creciente en el tiempo y es la referencia necesaria para otros campos.

El proceso de ordenamiento es conocido por tener complejidad numérica del orden O(NlogN), el costo es inevitable, sin embargo no es alto y además afecta solamente el proceso de compresión, no afecta la velocidad a la hora de la descompresión pues los datos, mientras se descomprimen, se entregan ordenados en el tiempo sin necesidad de regresarlos a su orden original, ordenar los registros temporalmente permite entre otras cosas indexar los datos de manera que se puedan hacer búsquedas (queries) por intervalos de tiempo de manera eficiente.

En este caso se consideran campos temporales los campos: *data_hora_chamada* e *id_data_carga*.

El método propuesto para compactar el campo *data_hora_chamada* en síntesis es el siguiente:

Truncar la hora de llamada a uno de los niveles: hora, media hora, cuarto de hora, decena de minutos, minuto, etc.

El campo truncado se puede codificar como una secuencia de enteros, ésta es no decreciente con intervalos constantes de longitud correlacionada con el valor de la hora que representa, el objetivo de este paso es aprovechar las frecuentes repeticiones en ciertos intervalos de tiempo para así reducir el tamaño del diccionario.

Se supone para ejemplificar que el campo *data_hora_chamada* está dado por truncamientos de la hora exacta en la cual se ha desechado la información de minutos y segundos, primeramente se codifican estas horas como números enteros que cuentan las horas a partir de una fecha cero de referencia.

A la hora de recuperar los datos se utiliza esta fecha inicial, que debe quedar almacenada en alguna parte; la longitud de los intervalos, horas en este caso; y la

información de la cantidad de intervalos transcurridos desde esta fecha, dada por la codificación de enteros.

La secuencia de valores del campo es entonces algo como:

1,1,1,2,3,3,3,3,3,3,3,4,4,4, etc.

Una forma de comprimir esta información es escribir la anterior secuencia como pares (hora de la llamada, cantidad de llamadas)

(1,3),(2,1),(3,7),(4,3), etc.

Si la cantidad de los datos es lo suficientemente grande, las primeras coordenadas son redundantes puesto que aparecen estrictamente en orden creciente y consecutivo, salvo los intervalos donde no se hayan realizado llamadas. Por un momento se considera que hay llamadas a todas las horas, entonces la secuencia anterior se puede abreviar como

3,1,7,3, etc.

y la posición del número en la sucesión nos da la hora correspondiente. Si hubiera cierta hora en la que no se realizan llamadas, se completa la sucesión con valores 0. Así si la sucesión original es:

1,1,1,2,3,3,3,3,3,3,3,4,4,4,6,6, etc.

Se tiene

3,1,7,3,0,2, etc.

significando que el valor 5, no aparece.

Dado que los contadores como función del tiempo presentan una distribución altamente regular según las horas del día y los días de la semana, se considera que una estrategia de compactación puede basarse en separar los datos de horarios de las llamadas por grupos semanales y diarios. Así se puede pensar en una matriz en la cual se almacena información como cantidad de llamadas en esa unidad de tiempo, a esta matriz se le llama matriz de los contadores.

Se ha verificado que el residuo del truncamiento, es decir, la cantidad de segundos restantes distribuye uniforme idénticamente distribuido.

De esta forma para la compresión de *data_hora_chamada* se procede de la siguiente manera:

Se almacena hora inicial (fecha inicial).

Se define la resolución a la cual se desea truncar las fechas ya sea por horas, medias horas, cuartos de hora, minuto, etc. y se obtienen dos nuevos campos:

El de la unidad anterior y el residuo.

Para el primer campo se aplica contador. Es decir se construyen pares (índice de intervalo, número de llamadas en intervalo). Implícitamente cada intervalo corresponde a un tipo de día: laboral o festivo (festivo, sábado o domingo). Este índice del intervalo queda implícito en la posición de la matriz, como se explicó anteriormente.

Ejemplo: se ha decidido que los intervalos sean por hora y se tienen los siguientes datos:

Fecha y hora inicial	Truncamiento a horas	Residuo
12/1/2003 0:21:49	12/1/2003 0	21:49
12/1/2003 3:11:10	12/1/2003 3	11:10
12/1/2003 3:12:55	12/1/2003 3	12:55
12/1/2003 5:10:45	12/1/2003 5	10:45
12/1/2003 7:49:25	12/1/2003 7	49:25
12/1/2003 7:51:26	12/1/2003 7	51:26
12/1/2003 8:53:46	12/1/2003 8	53:46
12/1/2003 8:58:55	12/1/2003 8	58:55
12/1/2003 9:12:37	12/1/2003 9	12:37
12/1/2003 9:23:21	12/1/2003 9	23:21
12/1/2003 9:30:05	12/1/2003 9	30:05
12/1/2003 9:30:30	12/1/2003 9	30:30
12/1/2003 9:37:34	12/1/2003 9	37:34
12/1/2003 9:39:20	12/1/2003 9	39:20
12/1/2003 9:39:59	12/1/2003 9	39:59
12/1/2003 9:40:58	12/1/2003 9	40:58
12/1/2003 9:45:00	12/1/2003 9	45:00
12/1/2003 9:50:30	12/1/2003 9	50:30
12/1/2003 10:09:23	12/1/2003 10	09:23
12/1/2003 10:11:38	12/1/2003 10	11:38
12/1/2003 10:17:59	12/1/2003 10	17:59
12/1/2003 10:25:04	12/1/2003 10	25:04
12/1/2003 10:27:58	12/1/2003 10	27:58

…

Se define 12/1/2003 00:00:00 como la fecha inicial y las horas se cuentan a partir de ésta, al mismo tiempo el residuo se lleva a segundos.

Indice Intervalo	Residuo en segundos	
0	21*60+49	1309
3	11*60+10	670
3	12*60+55	775
5	10*60+45	645
7	49*60+25	2965
7	51*60+26	3086
8	53*60+46	3226
8	58*60+55	3535
9	12*60+37	757
9	23*60+21	1401
9	30*60+05	1805
9	30*60+30	1830
9	37*60+34	2254

9	39*60+20	2360
9	39*60+59	2399
9	40*60+58	2458
9	45*60+00	2700
9	50*60+30	3030
10	09*60+23	563
10	11*60+38	698
10	17*60+59	1079
10	25*60+04	1504
10	27*60+58	1678

…

El campo de los índices se transforma en los pares

(0,1), (3,2), (5,1), (7,2), (8,2), (9,10), (10,5),…

Esto se puede ver como la representación 'sparse' de una matriz (vector). La representación no ‘sparse’ sería que los contadores se encuentran en la posición correspondiente al índice. Así en el caso anterior es:

1,0,2,0,1,0,2,10,5,..

Luego para cada día existe un vector de conteo de frecuencias. Así si para el día siguiente es:

0,0,0,0,1,3,0,20,15,…

Se puede pensar en estos arreglos como que se tiene una matriz

```
[[ [[1,0,2,0,1,0,2,10,5,…]],
   [[0,0,0,0,1,3,0,20,15,…]],
…]]
```

donde las filas corresponden a un día y las columnas a un mismo intervalo.

Estos contadores ocupan un espacio dependiendo de su magnitud, la cual correlaciona altamente según el tipo de día y el intervalo. Para ahorrar espacio, no se emplea la misma cantidad de bits por intervalo por tipo de día, sino que se sigue un criterio semi-adaptativo según los valores de los mismos.

Dentro de la misma categoría de día, para cada intervalo se calcula el rango de valores contados (mínimo y máximo), se almacena luego el vector de los valores mínimos por cada intervalo de tiempo, para cada tipo de día y luego se almacena el valor del redondeo por arriba del logaritmo de la diferencia entre máximo y mínimo +1. Este valor es la cantidad de bits que se utiliza para almacenar la diferencia entre el contador en cuestión y el valor mínimo para ese período para ese intervalo de tiempo.

Para el segundo campo (el residuo):

1309, 670, 775, 645, 2965, 3086, 3226, 3535, 757, 1401, 1805, 1830, 2254, 2360, 2399, 2458, 2700, 3030, 563, 698, 1079, 1504, 1678

Se tiene que deben existir punteros sobre los comienzos de los intervalos de manera que al realizar búsquedas (queries) sobre los mismos se pueda acceder rápidamente. Estos puntos se han marcado con letras negritas. Las solicitudes nunca se realizan sobre períodos de tiempo de intervalos menores que la unidad según la resolución que se ha tomado.

Partiendo de lo anterior, dentro de cada intervalo de tiempo se almacenan, no los valores sino, el primer valor y las diferencias de cada uno con respecto al anterior, esto aumenta grandemente las repeticiones de valores menores, luego estas diferencias se codifican con Huffman. Visto en el ejemplo tenemos:

1309, **670**, 5, **645**, **2965**, 121, **3226**, 309, **757**, 644, 404, 25, 424, 106, 39, 59, 242, 330, **563**, 135, 381, 425, 174.

Se reitera que en este caso aplicar esta técnica de diferenciación no es una pérdida, puesto que se aplica por intervalos menores a la resolución de los intervalos y por tanto las futuras llaves de acceso a los datos dan la información necesaria para recuperarlos, sin necesidad de regresar en el tiempo a descomprimir datos innecesarios.

Resumiendo en el código se almacena: la resolución, la hora inicial, el vector de bits por intervalo por tipo de día, el mensaje correspondiente a los contadores, y el diccionario y el mensaje correspondiente a los residuos

Para la compresión del otro campo temporal, es decir *data_carga* se propone:

Calcular días de diferencia entre día de *data_carga* y el día correspondiente a *data_hora_chamada* y aplicar Huffman canónico.

La serie anterior se puede volver a mejorar codificando la diferencia entre valores consecutivos.

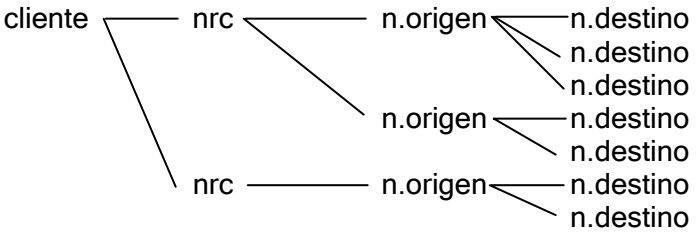
2.2.2 Compactación de campos conceptuales.

Por campos conceptuales se entiende aquellos cuyos valores tienen una relación de orden estructural con otros, en el ejemplo, serían los campos *id_cliente, nrc, id_localidade_origem, numero_origem, id_localidade_destino, numero_destino*, *id_carrier, id_subsegmento_cliente,* etc.

Como primer factor y más importante aquí está el cliente como fuente de decisión, distintos tipos de clientes pueden tener diferente comportamiento al decidir sobre el tipo de llamada. La categoría que se le da a un cliente queda almacenada físicamente en lo que se llama la ficha del cliente que recoge características estáticas y dinámicas del mismo.

La estructura semi-jerárquica de los campos conceptuales permite tomar decisiones basadas en las características del cliente. Se tienen llamadas, que pasan a través de líneas (*nrc*), el cliente tiene un conjunto de líneas por las que se comunica, por cada línea se realizan llamadas desde uno o más números de origen, cada número de origen se comunica a través de una sola línea, los números de origen llaman a distintos números destinos.

Estas relaciones están representadas en el siguiente esquema:



Al ordenar los registros en el tiempo, el campo *id_cliente*, al igual que los otros campos, queda también reordenado, este campo se compacta de manera global respetando este orden temporal, es importante tener presente que este campo es el que sirve de referencia (o puntero) a las fichas de los clientes para compactar y descompactar los demás campos conceptuales

Para la compresión de estos campos se realizan dos barridas, una en la que se crean las fichas de los clientes y una segunda en que se codifican los mensajes o registros según la misma.

Para la ficha del cliente la idea es definir una jerarquía en la que unos campos condicionan a otros de manera anidada, en el esquema obtenido para un cliente a partir de todas sus llamadas, algunos de los arcos deben ser obviados así como algunos de los nodos que estén duplicados, esta es la esencia de la propuesta.

Cada ficha de cliente tiene como primer parámetro un vector describiendo sus características lo que determina la estructura real de la ficha y cómo ésta debe ser leída a la hora de descomprimir.

Una de las características más importantes por incidir de manera decisiva en los factores de compresión que se deben obtener, es la que define la relación entre los campos *id_cliente, nrc, id_localidade_origem y numero_origem*, esto no es más que un trío que representa

(cantidad de *nrc* por *id_cliente*, cantidad máxima de *id_localidade_origem* por *nrc*, cantidad máxima de *num_origem* por *id_localidade_origem*).

Para una mejor visualización y comprensión en los siguientes ejemplos sólo se consideran los campos *id_cliente, nrc, y numero_origem*, de manera que la relación queda definida por un dúo y no por un trío.

Ejemplo 1:

498201 (104)
  278898661 (84)
    1115605223505 (66)
    1115635223505 (18)
  279031173 (20)
    1115635224348 (11)
    1115605224348 (9)

La primera columna representa el *id_cliente* 498201 que en este caso realiza 104 llamadas, (información ubicada entre paréntesis), la segunda columna representa los *nrc* de este cliente, en este caso: 278898661 con 84 llamadas y 279031173 con 20. La siguiente columna representa los números telefónicos de este cliente que se comunican por la correspondiente línea telefónica y el número de llamadas hechas por éstos.

Para utilizar este árbol como fuente de compresión de los registros, la idea es que en cada nivel del árbol, los elementos están ordenados de derecha a izquierda (de arriba hacia abajo en el esquema), por número de apariciones (llamadas). Luego el mensaje original se codifica según la posición que ocupa en el nodo correspondiente.

Para ver la aplicación de esta técnica basado en el ejemplo anterior, se supone que se tienen las llamadas:

Id_cliente, nrc, numero_origem, numero_destino
498201, 278898661, 1115605223505, 6200903146973
..
498201, 278898661, 1115635223505, 6700006342100
..
498201, 279031173, 1115605224348, 6200903146973
498201, 279031173, 1115635224348, 6200903146973
498201, 278898661, 1115605223505, 6700003911497
498201, 278898661, 1115605223505, 6727705461681

La codificación, es:

Id_cliente, nrc, numero_origem, numero_destino
498201, 1, 1

..
498201, 1, 2
..
498201, 2, 2
498201, 2, 1
498201, 1, 1
498201, 1, 1

Los puntos suspensivos indican que en el mensaje, las llamadas del mismo cliente no aparecen necesariamente una detrás de la otra, esta codificación se lee de la siguiente manera:

La línea
498201, 1, 1
quiere decir que esta llamada la realizó, el cliente 498201, por su línea más utilizada 278898661, con el teléfono más utilizado por esta línea 1115605223505.

Ejemplo 2:

Cliente: 106451 (104)
nrc: 4401747088 (51)
1100066310313 (51)
nrc: 1100499692 (46)
1100069671703 (46)
nrc: 3198363987 (4)
1100069558795 (4)
nrc: 3198363804 (3)
1100069541474 (3)

En este ejemplo el cliente es de tipo (4, 1), es decir, tiene 4 líneas y a través de cada una de ellas llama un sólo número origen, el cliente 106451 realizó 104 llamadas, a través de las líneas 4401747088, 1100499692, 3198363987 y 3198363804 donde la mayor cantidad de valores de numero_origem por línea es 1, luego es necesario enviar en el mensaje la información de qué línea se está utilizando.

Para ello se crea un diccionario de Huffman según sus frecuencias (codificación local) que para el ejemplo corresponde a:

nrc, (frecuencia), código asignado, (cantidad de bits)
4401747088, (51), 0, (1)
1100499692, (46), 11, (2)

3198363987, (4), 101, (3)

3198363804, (3), 100, (3)

Como este cliente no tiene líneas de múltiple *numero_origem*, no es necesario enviar al mensaje instrucción de cuál *numero_origem* se utilizó, en tiempo de descompactación, esta información se lee directamente de la ficha.

Hasta aquí la ficha de este cliente es algo como:

Id_cliente = 106451

Tipo cliente: (4,1)

Tipo de diccionario nrc: local

nrc, lengthbitshuffmancanonico, numero_origem:

4401747088, 1, 1100066310313

1100499692, 2, 1100069671703

3198363987, 3, 1100069558795

3198363804, 3, 1100069541474

Una estrategia parecida se utiliza para compactar los campos *id_localidade_destino* y *numero_destino*, denominada también como agenda del cliente.

Existen varios factores fundamentales en lo expuesto hasta aquí que aportan a la compresión:

- El árbol como estructura normalizadora pues al realizarse la construcción del árbol cada elemento es representado una y sólo una vez en la ficha del cliente.
- La utilización de códigos comunes con distintos significados según el contexto, en este caso el cliente.

Para cada uno de los otros campos se utilizan diversas estrategias que no es conveniente detallar aquí para no hacer tan extensa la exposición, considerando además que los campos claves y las ideas generales han sido ya expuestas, sólo parece meritorio repasar de forma muy breve la estrategia seguida para el campo id_carrier por su aplicación a otros campos.

La idea central queda descrita en el siguiente proceso:

Se estudia el comportamiento del cliente y por cada línea se define si la misma hace uso de varios *id_carrier* o sólo uno, si el uso de *id_carrier* es sencillo, se pasa el identificador de uso sencillo y el valor usado a la ficha, si el *id_carrier* es múltiple se pasa el identificador de uso múltiple y los valores con sus correspondientes longitudes de los códigos asignados por Huffman.

La información anterior puede aún ser abreviada, utilizando la ventaja de que la cantidad de valores de los *id_carrier* usados por cada línea es muy pequeña, esto tiene que ver ya con la compresión de la propia ficha.

Lo que se hace es crear diccionarios globales de *id_carrier* de la siguiente manera:

1) Se parte de una lista vacía de diccionarios globales.
2) Por cada línea nrc del cliente, se construye el diccionario de uso de los *id_carrier* según su frecuencia, este viene a estar dado por una tabla donde en la primera columna aparece el valor del *id_carrier*, ordenados de menor a mayor (para garantizar unicidad), y en la segunda la cantidad de bits asociados por Huffman.
3) Se busca en la lista de diccionarios globales comparando con los ya existentes, si no es igual a ninguno de ellos, se incluye en la lista, donde se le asocia la etiqueta correspondiente a su posición en la lista de diccionarios, si está en la lista de los diccionarios existentes, se coloca en la ficha del cliente la etiqueta correspondiente a dicho diccionario.

2.2.3 Estructura general del proceso.

Hasta ahora se ha explicado cómo los distintos campos se van codificando condicionados a otros campos. Es bueno aclarar que en el fichero final quedan entonces por un lado los registros donde los campos aparecen de manera secuencial, sin separadores, registro tras registro, y por otro lado las fichas de los clientes y demás diccionarios necesarios para decodificar las fichas o campos.

Dado que el cliente es el nodo principal en las relaciones de dependencia entre los distintos campos, esta información queda almacenada como primer elemento de cada registro, como se vio en el ejemplo anterior, por supuesto que nos referimos a su código según Huffman.

La propuesta consiste en ordenar los registros en el tiempo, dividir el mismo en intervalos, hacia los cuales existen punteros, relativo a este puntero se ponen otros punteros que indican el comienzo de los registros deseados, esto es particularmente útil para leer el comienzo del registro, reconocer el cliente, y a partir de ahí decidir futuras acciones.

Con los elementos antes expuestos, se puede ahora comprender la estrategia que se sigue durante el proceso de descompactación, lo que por supuesto rige la estructura que tiene el fichero con la información compactada, en síntesis es:

1) Se descompacta la hora

2) Se accede al registro deseado
3) Se pasa a leer dentro del mensaje codificado cuál es el cliente
4) Se lee su ficha
5) A partir de ésta entonces se descompactan los otros campos.

Para lograr este acceso aleatorio a un registro particular se propone la siguiente estructura para el fichero compactado, que será explicada a grueso modo a continuación, aclaremos que para abreviar, cuando se dice *tiempo* se refiere al campo *data_hora_chamada*, es decir año, mes, día, hora, minuto y segundo.

Se dividen los campos en dos bloques lógicos (no físico): *tiempo* y *señal* (otros campos).

Los registros se ordenan por el *tiempo* y se colocan puntos de sincronización de manera adecuada para facilitar el acceso aleatorio, los puntos de sincronización y los bloques de *tiempo* se almacenan en un bloque del archivo que se denomina *TimeBlock***.**

$Sync_0$
B_0
….
$Sync_k$
B_k

Los puntos de sincronización contienen la siguiente información:

time: tiempo de inicio de bloque.

offsetReg: desplazamiento del registro que le corresponde dentro del bloque de los registros compactados.

number: número de registros en el bloque.

Los bloques de *tiempo* de un punto de sincronización a otro se diferencian y se compactan como se explicó anteriormente.

Otro de los bloques del fichero es el de los registros compactados, este incorpora un nuevo campo por cada registro que cuenta la cantidad de bits que ocupa la *señal* del registro, de esta forma se pueden saltar registros sin necesidad de acceder a la ficha del cliente y descomprimir la *señal*.

Hay un bloque llamado *TimeIndex* que no es más que el índice de los puntos de sincronización, con esto se logra el acceso aleatorio de acuerdo con el *tiempo* de la consulta que se solicite, este contiene la siguiente información:

time: el tiempo indexado.

offBlock: desplazamiento del punto de sincronización, dentro de *TimeBlock* que contiene el tiempo indexado.

$Time_0$	0
$Time_1$	$offset(B_1)$
….	
$Time_k$	$offset(B_k)$

A partir de aquí un registro de la tabla original puede indexarse mediante el par $< offset(B_i), m >$, donde m es la posición en la lista dentro de *TimeBlock*.

2.3 Descripción de la implementación del compactador.

Una vez explicada la propuesta de compactación, se revisan ahora los aspectos que se consideran de mayor interés en la implementación del compactador de las tablas de llamadas telefónicas.

Compresión del campo *data_hora_chamada*

La compactación de este campo es clave en el desarrollo del compactador y en la obtención de los resultados, se sigue con exactitud la propuesta mencionada al principio de este informe, tanto para la conformación y almacenamiento de la matriz de los contadores del número de llamadas por intervalo de tiempo según el tipo de día, así como para el tratamiento de los restos, o sea, su diferenciación, estrategia de almacenamiento, métodos de acceso y codificación.

Se logran excelentes resultados tanto en el factor de compresión como en cuanto a las velocidades de compresión y descompresión, además se logra parametrizar la resolución o partición del tiempo lo que permite obtener la más adecuada para la inserción de los índices a la información de los demás campos contenidas en los registros, con vistas a lograr el mejor equilibrio entre factor de compresión y tiempo de descompresión.

Ya desde el primer campo se hace necesario desarrollar dos conjuntos de herramientas que son imprescindibles también para la implementación de todos los demás campos y que se detallan a continuación.

2.3.1 Lectura y escritura a nivel de bit.

Como se sabe los lenguajes de programación ofrecen métodos para la lectura y escritura a nivel de byte, pero para lograr una verdadera optimización del espacio en disco

y por tanto factores de compresión elevados se necesita que este acceso sea a nivel de bit. El conjunto de funciones que permite llevar a cabo esta tarea fueron implementadas de forma satisfactoria.

Luego de estudiar variantes ya existentes para este fin [MOF94], se desarrolla una herramienta propia que satisface todos los requerimientos lográndose así un conjunto de componentes llamados *cca_bitstream*, que permiten acceder a nivel de bit en modo de lectura o escritura ya sea en memoria o archivo, resolviéndose además el acceso aleatorio dentro de los ficheros de bits y lográndose también un manejo de caché más adecuado.

Así por ejemplo, un bitstream de lectura en memoria se define por la siguiente estructura:

```
typedef struct
{
    /** base address of buffer */
  cca_byte_t *base;
    /** position of next byte to read */
  cca_byte_t *pos;
    /** size of buffer */
  cca_int_t size;
    /** number of bytes remaining in buffer */
  cca_int_t remaining;
    /** current byte in process */
  cca_byte_t byte;
    /** number of bits unprocessed in byte */
  cca_byte_t btg;
    /** internal flags */
  int flags;
} cca_ibitmem_t;
```

Y algunas de las operaciones implementadas sobre ella son:

```
cca_ibitmem_t *cca_ibm_init(cca_ibitmem_t *bs,
                            cca_byte_t *base, cca_int_t size);
cca_byte_t cca_ibm_getbit(cca_ibitmem_t *bs);
cca_word_t cca_ibm_getword(cca_ibitmem_t *bs, cca_byte_t *l);
cca_byte_t cca_ibm_eos(cca_ibitmem_t *bs)
```

```
cca_mpos_t cca_ibm_tell(cca_ibitmem_t *bs)
void cca_ibm_seek(cca_ibitmem_t *bs, cca_mpos_t pos)
```

2.3.2 Implementación del algoritmo Huffman canónico.

En este se comienza haciendo una revisión de algunas implementaciones existentes de este algoritmo como por ejemplo la contenida en *shuff* [http://www.cs.mu.oz.au/], un programa que codifica y decodifica un fichero de enteros sin signo usando Huffman canónico.

Las implementaciones revisadas pueden ser perfectamente utilizadas en la compactación de los campos temporales, pero no responden a los requerimientos para la codificación de los campos conceptuales, donde se hace necesario por ejemplo el trabajo con varios diccionarios a la vez y un mayor control de los mismos.

Se decide por tanto, al igual que en el punto anterior, la elaboración de una versión propia que fue implementada obteniéndose así una componente llamada *cca_mrcoding* que realmente satisface las necesidades [PER05].

Esta componente define estructuras de datos y funciones asociadas para lograr la codificación y decodificación usando un Huffman canónico, basado en los trabajos de Alistar Moffat y Andrew Turpin [MOF02], [MOF01], además de otras ventajas como el uso de múltiples diccionarios de manera simultánea.

Así por ejemplo, la estructura de datos necesaria para mantener el estado del codificador es:

```
typedef struct
{
    /** alphabet size. */
  cca_size_t num_codes;
    /** codewords length assigned to each symbol. */
  cca_byte_t *clens;
    /** minimun length assigned. */
  cca_byte_t min_cw_len;
    /** maximun length assigned. */
  cca_byte_t max_cw_len;
    /* frequencies of each l-bit codeword. */
  cca_size_t cwlen_freq[CCA_MR_MAX_BITS+1];
    /** for each codeword length, the the first l-bit codeword */
```

```
  cca_word_t base[CCA_MR_MAX_BITS+1];
    /** the symbol number to the first of the l-bit codewords */
  cca_symbol_t offset[CCA_MR_MAX_BITS+1+1];
    /** number of precomputed codewords. */
  cca_size_t precomp_size;
    /** precomputed codewords. */
  cca_word_t *precomp_codes;
  cca_byte_t flags;
} cca_mr_coder_t;
```

Y algunas de las operaciones implementadas sobre ella son:

```
void cca_mr_encode(cca_mr_coder_t *coder,
                   cca_symbol_t symbol,
                   cca_word_t &cw,
                   cca_byte_t &l);
int cca_mr_encode_mem(cca_mr_coder_t *coder,
                      cca_obitmem_t *bs,
                      cca_symbol_t symbol);
```

Ya sobre esta base se desarrolla también una clase que permite codificar cualquier cadena de símbolos de cualquier tipo, extrayendo para esto por supuesto la frecuencia de aparición de cada uno de ellos, estableciendo una correspondencia entre los símbolos y sus índices dentro del arreglo de frecuencias ordenadas descendentemente que es lo que realmente se comprime [TUR00], a la vez se necesita compactar el propio diccionario que incluye la lista de símbolos y la información necesaria para lograr la persistencia del mismo, de manera que en tiempo de descompactación se puede reconstruir toda la información necesaria.

2.3.3 Algoritmo de ordenamiento.

Como ya se explicó anteriormente, la primera fase en la programación del compactador es el ordenamiento de los registros. Si la información a ordenar cupiera en la memoria RAM se pudiera aplicar el qsort de C, pero dado que la cantidad de registros de estas tablas es bastante grande se hace necesaria la implementación de una rutina de ordenación de datos a gran escala.

Después de analizar diversos algoritmos existentes para llevar a cabo esta tarea, [KNU73], se decide utilizar la biblioteca STXXL [http://www.stxxl.sourceforge.net/], en la

cual aparece implementado un algoritmo de ordenamiento equivalente al std::sort de la biblioteca STL del C++ pero para memoria externa, que es exactamente lo que se necesita.

Este algoritmo se aplica a la estructura de datos *vector* contenida también en esta biblioteca, que no es más que una estructura que implementa un arreglo sobre memoria externa y que permite el acceso aleatorio a sus elementos, la semántica de los métodos básicos del *vector* son también compatibles con el *vector* de la biblioteca STL.

Los tiempos de las operaciones de inserción y de acceso aleatorio sobre los vectores STXXL son del orden O(1), la configuración interna del vector como es el consumo de memoria interna (caché) utilizada es fácilmente controlada, esta clase posee un gran número de miembros que junto a los algoritmos implementados sobre este contenedor en la biblioteca le dan gran flexibilidad.

Estas y otras ventajas de la biblioteca como:

- Soporte de discos paralelos de manera transparente al usuario.
- Superposición de operaciones de E/S con los cálculos internos en muchos algoritmos y estructuras de datos.

llevan a su definitiva adopción y es la que soporta la implementación en el estado actual de su desarrollo, obteniéndose resultados satisfactorios en cuanto al tiempo requerido para llevar a cabo el ordenamiento.

Para ilustrar el uso de esta biblioteca, se muestran algunos fragmentos de la implementación del compactador:

```
#include <stxxl.h>
struct sentry
{
  long client;
  …
  sentry() {}
  …
};
struct sortentrytime
{
  bool operator() (const sentry & a, const sentry & b) const
  …
};
```

```
typedef stxxl::VECTOR_GENERATOR<sentry,BLOCKS,PGENTRY>::result
        vector_entry_t;
vector_entry_t ventry;
void read_entries()
{
  sentry entry;
  printf("\nreading entries...");
  …
  ventry.push_back(entry);
}
void sort_timeentries()
{
  printf("\n\nsorting registers by time...");
  stxxl::sort(ventry.begin(), ventry.end(), sortentrytime(),
              SORTMEM);
}
```

Compresión del campo *id_datacarga*.

Se logra siguiendo exactamente la propuesta sin ninguna dificultad y se obtienen también magníficos resultados en cuanto a factores de compresión y tiempos de compresión y descompresión.

Ficha del cliente. Campos *id_cliente, nrc, id_localidadeorigem, numero_origem*.

La ficha del cliente constituye un elemento clave en la concepción del compactador y por tanto merece especial cuidado durante su programación, en lo fundamental se siguió lo que se explicó en la propuesta, tanto lo referido a la confección de la ficha como tal, así como a la codificación de los mensajes de los registros asociados con cada caso, obteniéndose los resultados esperados.

2.3.4 Manipulación de archivos de grandes dimensiones.

La necesidad de poder leer ficheros fuentes extremadamente grandes, unido a la de poder direccionar ficheros binarios de gran tamaño [WIT94], motiva a la definición e implementación de funciones propias para la lectura y escritura en archivos, obteniéndose como resultado la biblioteca de funciones *cca_file*.

Esta define estructuras y métodos para la manipulación de archivos de grandes dimensiones, permitiéndose desplazamientos grandes dentro de éstos y acceso a los

mismos de forma aleatoria, manteniendo además un “buffer” interno en memoria para el acceso más rápido a la información.

Modelo de agenda. Campos *id_localidade_destino, numero_destino*.

La propuesta que se desarrolla aprovecha la naturaleza real del problema representado en la tabla, por lo que el análisis de los datos se hace por filas y no por columnas, es decir analizando las relaciones y/o dependencias entre los campos, de aquí que las estructuras a usar tengan gran nivel de anidamiento, lo que unido a la gran cantidad de registros a procesar hacen que las mismas crezcan extraordinariamente en la medida que se avanza un nivel en la cadena de dependencias.

2.3.5 Arbol de búsqueda.

Dada la necesidad de poder acceder a los datos de la ficha de un cliente en particular de forma rápida y totalmente aleatoria, y teniendo en cuenta que muchas de éstas no pueden albergarse completamente en la memoria RAM, se recurre a una estructura de datos que resuelve de forma eficiente el problema de las búsquedas y que a la vez trabaja sobre memoria extendida, el *btree.*

Después de revisar algunas implementaciones existentes como:

- la contenida en la biblioteca STXXL, donde aparece implementado como el contenedor *map,* muy similar al de la STL
- la que se encuentra en el paquete BDB (Berkeley DB) [http://www.sleepycat.com/] junto a otras estructuras como colas (*queue*) y tablas *hash*
- la implementada en la biblioteca Libgist [http://gist.cs.berkeley.edu/software/]

se decide utilizar ésta última, que es una implementación del árbol de búsqueda generalizado o GiST (Generalized Search Tree) [KOR99].

La clase central para los programadores de aplicaciones con libgist es *gist_m*, el método:

```
gist_m::create(filename)
```

crea un fichero de índices vacío y conecta el objeto *gist_m* al fichero.

Un GiST soporta los mecanismos estándares de actualización, inserción y eliminación de llaves, un nuevo par (*key*, *data_pointer*) se inserta con:

```
gist_m::insert(key, keylen, data, datalen)
```

Las entradas son borradas con:

```
gist_m::remove(query)
```

el cual requiere una “query” que describe las llaves de los elementos a eliminar.

Libgist soporta búsquedas con la bien conocida interfase de iteradores o cursores, las búsquedas se realizan con el método:

```
gist_m::fetch(key, keylen, data, datalen, eof).
```

habiendo establecido anteriormente el cursor asociado a una “query” a través de:

```
gist_m::fetch_init(cursor, query)
```

La extensión del btree implementada en la biblioteca, tiene dos clases: *bt_query_t*, la cual define los operadores de solicitudes para los *btree* (los usuales =, <, <=, >, >= y between), and *bt_ext_t*, la cual es la clase extensión para los *btree*.

Para ilustrar el uso de la biblioteca, se muestra aquí un ejemplo sencillo, en el cual se insertan dos enteros con sus datos y entonces se buscan todas las entradas en el intervalo [4, 6].

```
#include <gist.h>
gist_m bt_index;
bt_index.create("btree-file", &bt_int_ext);
int key1 = 5;
int data1 = 1;
int key2 = 3;
int data2 = 2;
bt_index.insert((void *)&key1, sizeof(int),
                (void *)&data1, sizeof(int));
bt_index.insert((void *)&key2, sizeof(int),
                (void *)&data2, sizeof(int));
bt_query_t q(bt_query_t::bt_betw, new int(4), new int(6));
gist_cursor_t cursor;
bt_index.fetch_init(cursor, &q);
bool eof = false;
int key, keysz, data, datasz;
while (!eof) {
  bt_index.fetch(cursor, key, keysz, data, datasz, eof);
  if (!eof) // process key and data...
}
```

Compresión del campo *id_subsegmento_cliente*.

Este campo se condiciona a la línea, se implementan tres variantes, o sea subsegmento estable, no estable y el caso de un sólo cambio obteniéndose buenos resultados.

Compresión del campo *id_carrier*.

Este campo se condiciona también a la línea y se implementan todos los casos posibles, también se implementa la lista global de diccionarios de *id_carrier*, aspecto que propicia obtener resultados excelentes en el factor de compresión para este campo, además esta misma estrategia se aplica en la compactación de otros campos.

Compresión del campo *id_old_client*.

En este caso el condicionamiento es al campo *id_cliente*, se tiene en cuenta la variante más común a la que se le da un tratamiento especial, no hubo dificultad alguna en su implementación y los resultados son excelentes.

Compresión de los campos *id_critica, id_tipo_trafego, id_degrau_tarifario, id_tipo_chamada*.

En este caso se sigue un esquema parecido al del *id_carrier*, sólo que se subordina al número destino y no se implementan por separado sino que se consideran las cuartetas posibles, aquí también se implementa un diccionario global para estas cuartetas obteniendo magníficos resultados.

Compresión del campo *segundos_duracaoreal*.

Para la compresión de este campo se utiliza una estrategia parecida al campo *data_carga.*

2.4 Innovaciones introducidas en la modelación con TOL.

El objetivo del siguiente tópico es el de describir las innovaciones que se introducen en la modelación para resolver un problema similar en algunos aspectos al de la distribución en prensa, así como algunos primeros enfoques en la resolución del problema desde el punto de vista de la modelación jerárquica/espacio de estados (o redes bayesianas) con información a priori [BOL94], [BOL96].

A los programadores en TOL ya se les había presentado esta problemática a la hora de hacer otros modelos con características parecidas y se implementó un conjunto de funciones llamadas BLR (Bayesian Linear Regression) [PER06], estas funciones dieron un paso importante en la comprensión e interpretación de modelos de este tipo, así como las líneas generales a seguir para su solución.

A continuación se describe brevemente en que consiste el problema que se debe resolver y la solución encontrada, se aprovecha para introducir algunos conceptos y métodos necesarios en la compresión del tema y se aclaran más las ideas del problema de prensa que es el que realmente motiva la realización del trabajo.

2.4.1 Regresión lineal general con estructura jerárquica en los parámetros.

Supongamos que se tiene una regresión lineal de la forma $Y = X\beta + E$ donde Y es el vector output, β es un vector de parámetros y E se distribuye como $N(0,\sigma^2 I)$, y que existen relaciones entre los parámetros ligados por diferentes estructuras probabilísticas. A su vez estas estructuras pueden poseer parámetros (hiper-parámetros) que a su vez pueden estar ligados con otros, estas estructuras que ligan los parámetros se llaman jerarquía de parámetros, los parámetros pueden relacionarse a diferentes niveles dentro de la jerarquía, esto determina la profundidad de la misma, lo podemos visualizar en el siguiente ejemplo:

Supongamos que se tiene el siguiente modelo:

$y = \mu_1 x_1 + \ldots + \mu_{10} x_{10} + e$ donde se tiene que $e \therefore N(0,\sigma^2 I)$,

$y, x_1, \ldots, x_{10}$ son vectores de dimensión n y supongamos que se tienen las siguientes relaciones:

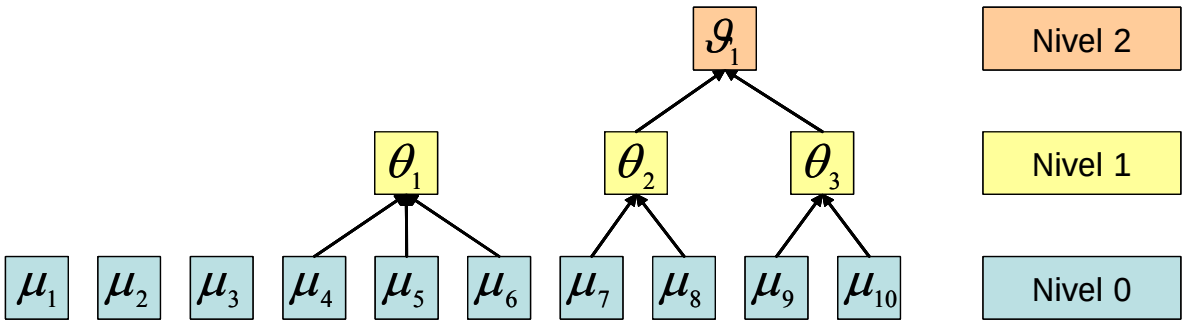
μ_1, μ_2, μ_3 son parámetros a estimar.

μ_4, μ_5, μ_6 son realizaciones independientes de $N(\theta_1, \sigma_1^2)$ con parámetros no necesariamente conocidos.

μ_7, μ_8 son realizaciones independientes de $N(\theta_2, \sigma_2^2)$

μ_9, μ_{10} son realizaciones independientes de $N(\theta_3, \sigma_3^2)$ que tienen la peculiaridad de que θ_2, θ_3 son realizaciones independientes de $N(\vartheta_1, \sigma_4^2)$

A modo de grafo queda como en la figura siguiente:



El anterior modelo se puede escribir como una regresión lineal extendida con las nuevas variables de los diferentes niveles que van añadiendo nuevas ecuaciones a las que ya se tienen [GOH02].

$$
\begin{aligned}
y &= \mu_1 x_1 + \mu_2 x_2 + \mu_3 x_3 + \mu_4 x_4 + \mu_5 x_5 + \mu_6 x_6 + \mu_7 x_7 + \mu_8 x_8 + \mu_9 x_9 + \mu_{10} x_{10} + e \\
0 &= \mu_4 - \theta_1 + e_{\theta_1} \\
0 &= \mu_5 - \theta_1 + e_{\theta_1} \\
0 &= \mu_6 - \theta_1 + e_{\theta_1} \\
0 &= \mu_7 - \theta_2 + e_{\theta_2} \\
0 &= \mu_8 - \theta_2 + e_{\theta_2} \\
0 &= \mu_9 - \theta_3 + e_{\theta_3} \\
0 &= \mu_{10} - \theta_3 + e_{\theta_3} \\
0 &= \theta_2 - \vartheta_1 + e_{\vartheta_1} \\
0 &= \theta_3 - \vartheta_1 + e_{\vartheta_1}
\end{aligned}
$$

Donde cada uno de los errores son de la forma $e_{\theta_1} \therefore N(0, \sigma_1^2 I_3)$, $e_{\theta_2} \therefore N(0, \sigma_2^2 I_2)$, $e_{\theta_3} \therefore N(0, \sigma_3^2 I_2)$ y $e_{\nu_1} \therefore N(0, \sigma_3^2 I_2)$.

Más adelante se describen algoritmos para el ensamblaje de esta matriz global inducida.

Finalmente la estimación del modelo se realiza mediante un método de muestreo Gibbs-Sampler [CAS92], este permite simular la verosimilitud conjunta de los parámetros, dadas las observaciones y las a priori sobre los parámetros, en cada paso se simula un bloque de parámetros condicionando al resto de los mismos, los cuales se dan como conocidos.

2.4.2 Los modelos empleados en la distribución.

La experiencia previa con la implementación de modelos lineales jerárquicos resueltos utilizando la metodología de regresión aumentada, se ve frustrada en el caso de la distribución de prensa por la hiper-dimensionalidad de este problema [ART07].

En breve las dimensiones para el caso de prensa son:

T = 365*6 = 2190, dimensión de las observaciones por kiosco.

N = 3000, número de kioscos mínimo. (número de nodos de observaciones)

P = 200, número de parámetros por kiosco.

Otra limitante del enfoque explicado es el modo de relacionar los parámetros, ocurre que las ventas entre los kioscos están relacionadas de varias maneras: transferencias de ventas entre ellos, patrones semanales similares, reacción a factores externos en proporciones parecidas, etc. Por esta razón estas ecuaciones son relacionadas de manera jerárquica en donde comportamientos parecidos de determinados parámetros se modelan por su agrupación en un nodo donde se define la relación entre los mismos. En sus versiones más sencillas las relaciones pueden ser:

- coeficientes comunes.

- coeficientes centrados en un valor.
- una regresión más general.

Ante esta situación se necesita encontrar un esquema que permita dar solución a este tipo de modelos que poseen una estructura jerárquica por naturaleza, y que no se vea limitado por la masividad de los mismos.

Los modelos empleados en prensa en cada kiosco pertenecen a la familia ARIMA con función de transferencia lineal, es decir la serie a analizar (output) se descompone en un filtro o parte explicada por variables exógenas y un ruido (noise) que se supone que sigue un proceso estocástico predecible mediante un modelo autorregresivo de media móvil.

La forma general de los modelos aplicados en prensa puede representarse mediante la siguiente fórmula:

$$t(y_t) = f_t + n_t$$
$$\phi(B)\nabla(B)n_t = \theta(B)a_t$$
$$a_t \underset{i.i.d.}{\therefore} N(0,\sigma)$$

donde:

- *t()* es una transformación de la familia Box-Cox. Actualmente sólo se emplean dos tipos de transformaciones sobre el output: logarítmica e identidad.
- y_t es la venta de periódicos, variable dependiente del modelo.
- f_t es el filtro o conjunto de efectos que explican exógenamente el comportamiento de la variable dependiente, está compuesto por funciones de transferencia lineales aditivas correspondientes a los impulsores y detractores de la venta.
- n_t es el ruido o parte de la venta no explicada por las variables exógenas, el ruido sigue un proceso ARIMA cuyos residuos (a_t) son ruido blanco gaussiano, es decir están idéntica e independientemente distribuidos como una normal de media cero y varianza constante en el tiempo (σ).

2.5 El modelo jerárquico lineal.

El objetivo de esta parte del trabajo es desarrollar las ideas de implementación para el bloque de los parámetros lineales de las regresiones de un modelo jerárquico, donde lo masivo de los datos requiere soluciones técnicas que optimicen el tiempo de ejecución de

la estimación, pero sobre todo tener la posibilidad de su manejo en memoria, en particular la solución está orientada al problema de prensa.

El primer paso que se lleva a cabo, por tanto, para analizar las ventas de periódicos es clasificar los puntos de venta en conjuntos con un comportamiento similar, en el tema de la clasificación se incorporan a TOL nuevas funciones "built-in" para la clasificación de vectores, la programación de las mismas está basada en la librería de código abierto de métodos de clasificación estadística llamada "The C Clustering Library" [http://bonsai.ims.u-tokyo.ac.jp/~mdehoon/software/cluster/software.htm#source]. En el capítulo 3 puede verse una explicación de las mismas.

2.5.1 Estructura computacional para representar el modelo.

A continuación se describe la estructura computacional adoptada para representar y manipular un modelo jerárquico lineal, en lo sucesivo HLM (Hierarchical Lineal Model). La representación se refiere a una caracterización de los elementos que la componen [PER07].

El objetivo es describir un modelo jerárquico lineal normal o HLM en términos más cercanos a la implementación computacional.

Un HLM normal es un modelo matemático que describe la relación lineal entre unas *variables observadas* y otras *variables explicativas* y a la vez se establecen sucesivas relaciones lineales entre *variables explicativas* y *variables latentes* no observadas. Además, a todas las variables no explicadas por una relación lineal se le especifica una distribución de probabilidad *«a priori»* la cual puede ser informativa o no informativa. Las relaciones de dependencias entre variables aleatorias observadas y no observadas se organizan por niveles como se describe a continuación.

El modelo explica un fenómeno que es medido en un conjunto de unidades de observación. Cada unidad se ha observado en un período de T intervalos de tiempo:

$$Y_i = X_i \cdot \beta_i^1 + u_i^1 \mid u_i^1 \sim N(0, \Sigma_i^1),\ i = 1, 2,..., N_1,\ Y_i = (y_{i1},..., y_{ir}) \qquad (1)$$

Las ecuaciones anteriores se denominan ecuaciones del nivel 1 o nivel observado. A partir de aquí se definen relaciones en las que las variables no observadas o parámetros del nivel 1 dependen linealmente de nuevas variables aleatorias ocultas:

$$\beta_j{}^{*1} = X_j^2 \cdot \beta_j^2 + u_j^2 \mid u_j^2 \sim N(0, \Sigma j^2),\ j = 1, 2, ..., N_2,\ \beta_j{}^{*1} = (\beta_{i1j1}{}^1 ,..., \beta_{irjs}{}^1) \qquad (2)$$

Las ecuaciones anteriores se denominan de nivel latente o no observado, dado que todas las variables aleatorias que intervienen en ella son no observadas.

Aunque en la práctica se suelen usar 2 o 3 niveles de jerarquía, es posible definir tantos niveles como se quiera:

$$\beta_j^{*k-1} = X_j^k \cdot \beta_j^k + u_j^k \mid u_i^k \sim N(0, \Sigma_j^k),\ j = 1, 2, ..., N_k,\ \beta_j^{*k-1} = (\beta_{i1j1}^{k-1}, ..., \beta_{irjs}^{k-1}) \qquad (3)$$

Además de las ecuaciones de nivel latente, tenemos las ecuaciones de información a priori las cuales definen una distribución de probabilidad para un vector aleatorio del modelo:

$$\beta^p \sim N(u^p, \Sigma^p) \qquad (4)$$

donde β^p es un vector de variables de modelo, u es un vector de medias conocido y Σ es una matriz de varianzas y covarianzas también conocida.

Se asume además que los vectores de errores μ_j^l de las unidades de un mismo nivel *l* son independientes entre sí: $E(u_i^l \cdot (u_j^l)^T) = 0,\ i \neq j.$

De forma general en el nivel de observación se define una distribución de probabilidad conjunta para las observaciones de cada unidad en función de variables aleatorias (ecuación 1) y en niveles sucesivos se definen distribuciones de probabilidad conjunta para variables aleatorias no observadas que aparecen en el nivel inmediato inferior en función de nuevas variables aleatorias no observadas (ecuación 3). Por último para aquellas variables aleatorias no condicionadas en los niveles latentes, se define una distribución de probabilidad multivariada normal de parámetros conocidos (ecuación 4).

Los problemas que se quieren resolver asociado a un HLM son: estimación/simulación de las variables aleatorias no observadas y previsión de nuevas observaciones condicionadas a los parámetros estimados. La estimación/simulación se hace condicionando el modelo a variables conocidas que pueden ser: observaciones en el nivel 1, parámetros dados como información a priori y covarianzas simuladas en un proceso externo.

El modelo aquí descrito puede ser visto como un caso particular de red bayesiana con las siguientes características:

- Las distribuciones de probabilidad son normales.
- Las variables del modelo se han agrupado en unidades.
- Las unidades se han agrupado en niveles según independencias de los errores.
- Los niveles se han ordenado y sólo pueden existir condicionamientos directos entre variables de un nivel y el nivel antecesor según el orden de niveles establecido.

La elección de las restricciones anteriores obedece a que por una parte el problema que se quiere resolver satisface esos requerimientos y por otra permite elaborar algoritmos eficientes de simulación de los parámetros desconocidos. Un enfoque basado en una red

bayesiana general [WIK07a], no sería capaz de explotar las suposiciones anteriores. En particular interesa la independencia condicional entre las unidades o grupos de unidades de un mismo nivel con el objetivo de diseñar algoritmos paralelos de simulación.

Al igual que con una red bayesiana, se utiliza una representación basada en grafos. Un modelo HLM normal es una estructura jerárquica multilineal que puede ser modelada mediante un *grafo acíclico dirigido* o *DAG*.

Un grafo es una estructura matemática que permite establecer relaciones entre los elementos de un conjunto. En este caso los elementos del conjunto son las distribuciones de probabilidad conjunta y las distribuciones de probabilidad *«a priori»*.

En un modelo HLM se pueden identificar 3 tipos de nodos: *nodos observados*, *nodos no observados* o *latentes* y *nodos informativos* o de información *«a priori»*.

Los *nodos observados* definen la relación lineal entre *variables aleatorias observadas* y *variables aleatorias no observadas* denominadas también *parámetros de regresión lineal*.

Los *nodos no observados* definen relaciones lineales de variables aleatorias no observadas (*condicionadas*) en función de otras variables aleatorias (*condicionantes*). Las *variables aleatorias condicionantes* se conocen como *variables latentes*. Las *variables aleatorias condicionadas* pueden ser tanto *parámetros* de los nodos observados como *variables latentes*.

Los *nodos informativos* definen una distribución de probabilidad *«a priori»* para un vector de variables aleatorias. La distribución de probabilidad dada en los nodos «*a priori*» está completamente determinada por parámetros conocidos.

A partir de aquí cuando se utilice el término *observación* se refiriere a una *variable aleatoria observada*. El término *variable* se emplea tanto para referirse a los *parámetros* en los *nodos observados* como a las *variables latentes*.

Numeración local y global.

Se denota por $N = \{N_1, ..., N_l\}$ al conjunto de nodos del grafo HLM, quedando implícito una correspondencia biunívoca entre N y el conjunto $\{1, \ldots, l\}$. De esta forma un nodo queda perfectamente determinado por su índice dentro de N.

Se denota además por $V = \{v_1, \ldots, v_n\}$ al conjunto ordenado de todas las *variables* del modelo.

En cada nodo se puede identificar dos conjuntos de variables: *condicionadas* y *condicionantes*. Las variables *condicionadas* pueden ser *observadas* o *no observadas*.

Cuando el conjunto de variables *condicionadas* son *observadas* este se denota como: $Y_i = \{y_{i1}, ..., y_{imi}\}$.

Si el conjunto de variables *condicionadas* son *no observadas* éste se denota como: $A_i = \{\alpha_{i1}, ..., \alpha_{imi}\} \subset V$.

En A_i existe un orden local dado por una correspondencia biunívoca entre A_i y $\{1, \ldots, m_i\}$. Dado que $A_i \subset V$ se puede definir una aplicación que establezca una relación entre el ordinal de los elementos de A_i y el ordinal de los correspondientes elementos en V:

$$I_{Ai}: \{1, ..., m_i\} \rightarrow \{1, ..., n\}$$

$$I_{Ai}(r) = s \Leftrightarrow \alpha_{ir} \equiv v_s \quad (5)$$

La aplicación I_{Ai} establece una correspondencia entre el *índice local* de una *variable condicionada* en A_i y su correspondiente *índice global* en V.

El conjunto de *variables condicionantes* en un nodo N_i, se denota por $\beta_i = \{ \beta_{i1}, ..., \beta_{iki}\} \subset V$. En β_i existe un orden local dado por una correspondencia biunívoca entre β_i y $\{1, \ldots, k_i\}$. A partir del orden local en β_i y el orden global en V se define la aplicación:

$$I_{Bi}: \{1, ..., k_i\} \rightarrow \{1, ..., n\}$$

$$I_{Bi}(r) = s \Leftrightarrow \beta_{ir} \equiv v_s \quad (6)$$

La aplicación I_{Bi} establece una correspondencia entre el *índice local* de una *variable condicionante* en β_i y su correspondiente *índice global* en V.

A partir del orden establecido en los elementos de V, A_i y β_i se definen también los siguientes vectores:

$$\beta = (v_1, ..., v_n)^T \quad (7)$$

$$\alpha_i = (\alpha_{i1}, ..., \alpha_{imi})^T \quad (8)$$

$$\beta_i = (\beta_{i1}, ..., \beta_{imi})^T \quad (9)$$

Dicha notación vectorial resulta útil a la hora de expresar las relaciones lineales definidas en los nodos del grafo.

Nodos observados.

Los nodos observados son los nodos del nivel 1. En este nivel se especifican las relaciones entre los parámetros del nivel de observación y las observaciones.

Definición 9. Un *nodo observado* es una 4-tupla $Ni := <Y_i, X_i, \beta_i, \Sigma_i>$ que cumple la siguiente relación probabilística*:*

$$Y_i \sim N(X_i \cdot \beta_i, \Sigma_i) \quad (10)$$

donde:

- $Y_i = (y_{i1}, ..., y_{imi})^T$ es un vector de *variables aleatorias observadas.*
- X_i *es* una matriz de *efectos observados o inputs* de dimensión $m_i \times k_i$.
- β_i *es* el vector de *variables explicativas no observadas.*
- Σ_i es la *matriz de covarianzas de las observaciones* de tamaño $m_i \times m_i$.

La relación probabilística 10 se puede escribir mediante una *ecuación aleatoria*:

$$Y_i = X_i \cdot \beta_i + \varepsilon_i \mid \varepsilon_i \sim N(0, \Sigma_i) \quad (11)$$

De un nodo observado N_i se puede conocer entonces:

Dimensión del vector de observaciones:

`Rows(i)` = m_i

Dimensión del vector de parámetros:

`Columns(i)` = k_i

Valor de una observación:

`Observation(i,j)` = $y_{ij} \mid 1 \leq j \leq m_i$

Valor de una celda de la matriz de inputs:

`InputValue(i,r,c)` = $X_i(r, c) \mid 1 \leq r \leq m_i,\ 1 \leq c \leq k_i$

Valor de una celda de la matriz de covarianzas:

`Covarianze(i,r,c)` = $\Sigma_i (r, c) \mid 1 \leq r \leq m_i,\ 1 \leq c \leq m_i$

Indice global de un parámetro de regresión local:

`VariableIndex(i,k)` = $I_{Bi}(k) \mid 1 \leq k \leq k_i$

Nodos latentes.

Los nodos latentes se definen por niveles a partir del nivel 2. En un *nivel latente* dado los nodos definen distribuciones de probabilidad normal para vectores de parámetros del nivel inferior. Dicha distribución se expresa en función de un vector media y una matriz de covarianza. El vector media es una combinación lineal de nuevos parámetros en función de una matriz de efectos conocida.

Definición 10. *Uno nodo latente de un nivel* $l > 1$ es una 4-tupla $Ni := <\alpha_i, X_i, \beta_i, \Sigma_i>$ que cumple la siguiente relación probabilística:

$$\alpha_i \sim N(X_i \cdot \beta_i, \Sigma_i) \quad (12)$$

donde:

- $\alpha_i = (\alpha_{i1}, \ldots, \alpha_{i\ mi})^T$ es un vector de *variables aleatorias no observadas.* Las componentes de α_i son variables aleatorias que aparecen como *variables condicionantes* en nodos del nivel l -1.
- X_i es una *matriz de efectos observados o inputs* de dimensión $m_i \times k_i$.
- β_i es el vector de *variables no observadas.*
- Σ_i es la *matriz de covarianzas del vector* α_i *de tamaño* $m_i \times m_i$.

La relación probabilística 12 se puede escribir mediante una ecuación aleatoria:

$$0 = X_i \cdot \beta_i - \alpha_i + \varepsilon_i \mid \varepsilon_i \sim N(0, \Sigma_i) \quad (13)$$

o equivalentemente,

$$0 = (X_i - I)\cdot(\beta_i \ \alpha_i)^T + \varepsilon_i \mid \varepsilon_i \sim N(0, \Sigma_i) \quad (14)$$

De un *nodo latente* N_i se puede conocer entonces:

Dimensión del vector de *variables latentes dependientes*:

`Rows(i)` = m_i

Dimensión del vector de parámetros o *variables latentes independientes*:

`Columns(i)` = k_i

Valor de una celda de la matriz de inputs:

`InputValue(i,r,c)` = $X_i(r, c) \mid 1 \le r \le m_i,\ 1 \le c \le k_i$

Valor de una celda de la matriz de covarianzas:

`Covarianze(i,r,c)` = $\Sigma_i (r, c) \mid 1 \le r \le m_i,\ 1 \le c \le m_i$

Indice global de una variable latente independiente:

`ObsVariableIndex(i,j)` = $I_{Ai}(j) \mid 1 \le j \le m_i$

Indice global de un parámetro de regresión local:

`VariableIndex(i,k)` = $I_{Bi}(k) \mid 1 \le k \le k_i$

Nodos «*a priori*».

Definición 11. Un *nodo con información «a priori»* es una terna $Ni := <\alpha_i, \mu_i, \Sigma_i>$ que define una distribución de probabilidad a priori para un vector aleatorio del modelo HLM*:*

$$\alpha_i \sim N(\mu_i, \Sigma_i) \quad (15)$$

donde:

• α_i es un vector de variables aleatorias para el cual se especifica la distribución de probabilidad «a priori»

• μ_i es la media del vector α_i

• Σ_i la matriz de covarianzas del vector α_i.

la relación 15 puede reescribirse como:

$$\mu_i = \alpha_i + \varepsilon_i \mid \varepsilon_i \sim N(0, \Sigma_i) \quad (16)$$

De esta manera para un nodo N_i con información *a priori* se puede conocer:

Dimensión del vector de *variables latentes dependientes*:

`Rows(i)` = m_i

Media de una componente del vector α_i:

`PriorMean(i,j)` = $\mu_i(j) \mid 1 \le j \le m_i$

Valor de una celda de la matriz de covarianzas:

`Covarianze(i,r,c)` = $\Sigma_i (r, c) \mid 1 \le r \le m_i,\ 1 \le c \le m_i$

Indice global de una componente del vector α_i:

`PriorVariableIndex(i,j)` = $I_{Ai}(j) \mid 1 \le j \le m_i$

Condicionamiento a las covarianzas.

La estimación/simulación de los parámetros en el HLM se hace condicionado a las matrices de covarianzas que son conocidas. Para un estado conocido de las matrices de covarianzas los sistemas nodales pueden trasformarse de forma tal que los errores se distribuyan según $N(0, I)$ y aplicar la teoría de regresión lineal. Si se escribe un sistema general como

$$Z = A\cdot \beta + \varepsilon \mid \varepsilon \sim N(0, \Sigma)$$

entonces la ecuación trasformada es:

$$Y = X \cdot \beta + \eta, \eta \sim N(0, I) \mid \Sigma = L\cdot L^T, Y = L^{-1} \cdot Z, X = L^{-1} \cdot A \quad (17)$$

las matrices L^1 y L^{-1} son triangulares inferiores.

De esta forma las ecuaciones en los nodos quedan:

Observación:

$$L_i^{-1} \cdot Y_i = L_i^{-1} \cdot X_i \cdot \beta_i + \eta_i \mid \eta_i \sim N(0, I), \Sigma_i = L_i \cdot L_i^T \quad (18)$$

en este caso se define una función para aplicar la transformación a la ecuación así como funciones de acceso a las magnitudes transformadas:

Aplicar la transformación: calcula las observaciones y la matriz de inputs transformados

`ApplyCovarTransformation(i)`

Valor de una observación transformada:

`TransfObservation(i,j)` $= (L_i^{-1} \cdot Y_i)(j) \mid 1 \le j \le m_i$

Valor de una celda de la matriz de inputs transformada:

`TransfInputValue(i,r,c)` $= (L_i^{-1} \cdot X_i)(r, c) \mid 1 \le r \le m_i, 1 \le c \le k_i$

Latente:

$$0 = (L_i^{-1} \cdot X_i - L_i^{-1}) \cdot (\beta_i \ \alpha_i)^T + \eta_i \mid \eta_i \sim N(0, I), \Sigma_i = L_i \cdot L_i^T \quad (19)$$

Análogamente se define la función que aplica la transformación y las de acceso a las magnitudes transformadas:

Aplicar la transformación: calcula las observaciones y la matriz de inputs transformados

`ApplyCovarTransformation(i)`

Acceso de una celda de la matriz de inputs transformada:

`TransfInputValue(i,r,c)` $= (L_i^{-1} \cdot X_i)(r, c) \mid 1 \le r \le m_i, 1 \le c \le k_i$

Acceso a la parte triangular inferior de matriz de transformación L_i^{-1}:

`LowerTransfValue(i,r,c)` $= L_i^{-1}(r, c) \mid 1 \le r \le m_i, 1 \le c \le r$

Informativo:

$$L_i^{-1} \cdot \mu_i = L_i^{-1} \cdot \alpha_i + \eta_i \mid \eta_i \sim N(0, I), \Sigma_i = L_i \cdot L_i^T \quad (20)$$

y las funciones:

Aplicar la transformación: calcula las observaciones y la matriz de inputs transformados

`ApplyCovarTransformation(i)`

Acceso al vector de medias transformado:

`TransfPriorMean(i,j)` = $L_i^{-1}\,\mu_i\,(j) \mid 1 \leq j \leq m_i$

Acceso a la parte triangular inferior de matriz de transformación L_i^{-1}:

`TransfLowerValue(i,r,c)` = $L_i^{-1}(r, c) \mid 1 \leq r \leq m_i,\ 1 \leq c \leq r$

A partir de esta suposición los subsistemas lineales asociados a los nodos pueden transformarse como paso previo al cálculo del aporte nodal. No obstante, los algoritmos siempre hacen explícita la transformación previa al acceso a los datos. Por temas de eficiencia y dado que la matriz de covarianza se mantiene constante es posible que la transformación se aplique sólo una vez lo cual quedará indicado.

Aristas del grafo.

Las aristas del grafo HLM vienen dadas implícitamente por las dependencias entre las variables definidas en los nodos. Informalmente, existe una arista con origen en un nodo C y destino en un nodo D, $C \rightarrow D$, si existen *variables condicionantes* en D que aparecen en C como *variables condicionadas*.

Está claro que los *nodos observados* no pueden ser origen de ninguna arista y que los *nodos latentes* y *nodos «a priori»* son origen de alguna arista.

Lo anterior se puede formalizar a partir de los conjuntos A_i de *variables condicionadas* y B_i de *variables condicionantes*. Se tiene entonces que:

- Si N_i es un *nodo observado* entonces, $A_i = 0 \wedge B_i \neq 0$, o en términos del grado del nodo: $dout(N_i) = 0 \wedge din(N_i) > 0$.
- Si N_i es un *nodo latente* entonces, $A_i \neq 0 \wedge B_i \neq 0$, o en términos del grado del nodo: $dout(N_i) > 0 \wedge din(N_i) > 0$
- Si N_i es un nodo *«a priori»* entones, $A_i \neq 0 \wedge B_i = 0$, o en términos del grado del nodo: $dout(N_i) > 0 \wedge din(N_i) = 0$.

Las aristas del grafo quedan definidas de la siguiente forma:

Definición 13. Dados dos nodos $N_i, N_j \in N$ cualesquiera del grafo HLM, se dice que existe una *arista* $e_{ij} = (N_i, N_j) \in E \Leftrightarrow A_i \cap B_j \neq 0$.

Como se ha visto sólo los nodos latentes y *«a priori»* pueden ser origen de aristas. Se denomina a éstos nodos *condicionantes*. Estos nodos pueden condicionar a más de un nodo hijo o *nodo condicionado* en el nivel inferior.

Se denota por $A_i \cap B_j = \{r_1^{ij}, \ldots, r_s^{ij}\}$ y $C_{ij} = \{1, 2, \ldots, |A_i \cap B_j|\}$. A partir de estos dos conjuntos se definen dos funciones que son básicas para definir la distribución condicional

completa de los parámetros de un nodo del grafo. Estas funciones definen una correspondencia entre los índices locales de los elementos de $A_i \cap B_j$ y los elementos de A_i y B_j respectivamente.

La primera de esas funciones identifica el índice de ecuación en el nodo padre N_i que condiciona a un parámetro del nodo hijo N_j dado por su índice local dentro del conjunto $A_i \cap B_j$:

$$\Gamma_{ij}^{a} : C_{ij} \rightarrow \{1,\ldots, |A_i|\} \qquad (21)$$

$$\Gamma_{ij}^{a}(k) = l \;\Leftrightarrow\; r_k^{ij} \equiv \alpha_{il}$$

La segunda identifica el índice local de una *variable condicionante* dentro del nodo hijo N_j y que es condicionada en el nodo padre N_i dado su índice local dentro del conjunto $A_i \cap B_j$:

$$\Gamma_{ij}^{b} : C_{ij} \rightarrow \{1,\ldots, |B_j|\}$$

$$\Gamma_{ij}^{b}(k) = l \;\Leftrightarrow\; r_k^{ij} \equiv \beta_{jl}$$

La información codificada en las funciones Γ_{ij}^{a} y Γ_{ij}^{b} pueden asociarse a cada una de las aristas del grafo. A partir de aquí se definen, desde un punto de vista algorítmico, las siguientes funciones asociadas a una arista $e_{ij} = (N_i, N_j)$:

Cardinalidad del condicionamiento:

`EdgeCard(i,j)` = $|A_i \cap B_j|$

Indice local de ecuación condicionante:

`EdgeEquationIndex(i,j,k)` = $\Gamma_{ij}^{a}(k) \mid 1 \le k \le |A_i \cap B_j|$

Indice local de parámetro condicionado:

`EdgeVariableIndex(i,j,k)` = $\Gamma_{ij}^{b}(k) \mid 1 \le k \le |A_i \cap B_j|$

2.5.2 Estimación puntual.

A continuación se dan descripciones algorítmicas necesarias en la estimación de estos modelos.

La estimación puntual consiste en encontrar un valor «óptimo» de las variables aleatorias no observadas en función de las observaciones, las matrices explicativas, covarianzas y la información a priori. Este «óptimo» se corresponde con el valor más probable según la distribución conjunta de todas las variables aleatorias no observadas del modelo condicionado a las anteriores magnitudes conocidas.

Las relaciones lineales que se establecen en los nodos del modelo, visto de forma conjunta, se pueden reescribir en un sistema global aleatorio lineal con errores

independientes y distribución normal de media y varianza conocida, véase [GEL03] capítulo 15:

$$Y_* = X_* \cdot \beta + \varepsilon \mid \varepsilon \sim N(0, \Sigma_*) \quad (22)$$

Condicionado a las covarianzas cada nodo del modelo puede verse según la ecuación tipo 17 de forma tal que el *sistema conjunto* de todas las ecuaciones definidas en cada nodo del grafo HLM se puede representar matricialmente mediante una relación lineal del tipo:

$$Y = X \cdot \beta + \eta \mid \eta \sim N(0, I) \quad (23)$$

Donde, en 23, la matriz X contiene el aporte de cada uno de los nodos según su matriz local X_i transformada y el vector Y se forma a partir del aporte de cada uno de los nodos según el término independiente de los sistemas 18, 19 y 20.

Matriz Global Inducida.

A pesar de que las ecuaciones aportadas por los diferentes tipos de nodos pueden verse de forma unificada conviene diferenciarlos por cada tipo ya que ello permite explotar la estructura *sparse* de las matrices en los *nodos latentes*.

Si se indexan las ecuaciones del sistema global 22 con los números $1, \ldots, m$ y se hace corresponder en orden a cada nodo N_i un índice sucesivo de ecuaciones se llega a un indexado global de las ecuaciones en los nodos.

Cada nodo N_i aporta un número de ecuaciones igual a m_i. De esta forma el nodo N_1 aporta las ecuaciones con índices $1, \ldots, m_1$; el nodo N_2, las ecuaciones $m_1+1, \ldots, m_1+m_2$ y así sucesivamente. De forma general el nodo N_i aporta las ecuaciones con índices $\sum_{1,i-1} m_j + 1, \ldots, \sum_{1,i} m_j$.

Para describir el aporte de los tipos de nodos al sistema global se asume que se cuenta con la función `EqBaseIndex(i)` que da el índice de base a partir del cual se aportan las ecuaciones del nodo N_i con índice i.

Aporte del *nodo observado.*

Los *nodos observados* aportan al sistema global un vector de observaciones denso y una matriz de "inputs" densa.

Sea N_i un nodo observado, la aportación de N_i al sistema global 22 está dada en el siguiente algoritmo:

```
base = EqBaseIndex(i)
ApplyCovarTransformation(i)
for r = 1 to Rows(i)
 Y(base+r) = TransfObservation(i,r)
```

```
 for k = 1 to Columns(i)
  c = VariableIndex(i,k)
  X(base+r,c) = TransfInputValue(i,r,k)
 end for
end for
```

Aporte del *nodo latente.*

Los *nodos latentes* aportan al sistema global un vector de observaciones nulo y una matriz de inputs sparse, ver ecuación 19 para el aporte condicionado a las covarianzas.

Sea N_i un nodo latente, la aportación de N_i al sistema global 22 está dada en el siguiente algoritmo:

```
base = EqBaseIndex(i)
ApplyCovarTransformation(i)
for r = 1 to Rows(i)
 for k = 1 to Columns(i)
  c = VariableIndex(i,k)
  X(base+r,c) = InputValue(i,r,k)
 end for
 for k = 1 to r
  c = ObsVariableIndex(i,r)
  X(base+r,c) = -TransfLowerValue(i,r,k)
 end for
end for
```

Aporte del *nodo «a priori».*

Los *nodos «a priori»* aportan al sistema global un conjunto de ecuaciones correspondientes al sistema 20, transformado como resultado del condicionamiento a las covarianzas.

Sea N_i un nodo *«a priori»*, la aportación de N_i al sistema global 22 está dada en el siguiente algoritmo:

```
base = EqBaseIndex(i)
ApplyCovarTransformation(i)
for r = 1 to Rows(i)
 Y(base+r) = TransfPriorMean(i,r)
 for k = 1 to r
  c = PriorVariableIndex(i,k)
```

```
    X(base+r,c) = TransfLowerValue(i,r,k)
   end for
  end for
```

Una solución del sistema global puede hacerse aplicando el método del Gradiente conjugado.

2.5.3 Simulación bayesiana.

La simulación bayesiana de las variables condicionadas a las observaciones y parámetros conocidos consiste en generar una muestra del vector β a partir de la distribución conjunta $\pi(\beta|Y_*,X_*,\Sigma_*)$. Bajo la suposición de normalidad se sabe que la distribución de β es normal:

$$\beta|Y_*,X_*,\Sigma_* \sim N(.,.) \qquad (24)$$

A partir de la ecuación 24 es fácil generar una muestra mediante métodos conocidos, pero en el caso de prensa, debido a las magnitudes del modelo, es imposible representar en un ordenador las matrices que aparecen.

Existen varios métodos para generar una muestra a partir de una distribución desconocida o intratable computacionalmente. Estos métodos generan una secuencia correlacionada que en el límite converge a la distribución objetivo [ROB05]. Para una distribución desconocida de la cual sólo se sabe evaluar una función proporcional a la densidad, con factor de proporcionalidad desconocido, se puede aplicar el método general *Metropolis-Hastings* [CHI95]. Si además se identifica un particionamiento de los parámetros de forma tal que se conoce un método para generar muestras para cada bloque de parámetros entonces se puede aplicar un caso particular del *Metropolis-Hastings* el cual tiene, de forma general, mejores propiedades de convergencia en la muestra conjunta. Este último método se conoce como *generador Gibbs* [CAS92], [ROB05].

En el caso del modelo HLM, el vector conjunto de parámetros queda particionado por los nodos del grafo. Este particionamiento favorece el mezclado de la cadena de Markov [KAA98] generada ya que tiende a muestrear en conjunto aquellos parámetros más correlacionados. De esta forma el vector conjunto de variables V a simular en el modelo se particiona en los bloques de variables B_i asociadas a los nodos: $V = \bigcup B_i$

También se escoge un orden de muestreo de los bloques determinista orientado a los niveles de forma tal que se simulen en orden primero los bloques del nivel de observación

y luego los sucesivos niveles latentes. En el siguiente fragmento se describe el flujo del *generador Gibbs* para el HLM.

β vector de variables particionado en bloques $(\beta_1^1,\ldots,\beta_{b1}^1,\ldots,\beta_1^L,\ldots,\beta_{bl}^L)$
$\beta^{(0)} = (\beta_1^{1(0)}, \ldots, \beta_{b1}^{1(0)}, \ldots, \beta_1^{L(0)}, \ldots, \beta_{bl}^{L(0)})$ valor inicial del Gibbs
for t = 1 to M
 $\beta^{(t)} = \beta^{(t-1)}$
 for l = 1 to L
  for j = 1 to b_l
   $\beta_j^{l(t)}$ = SampleFrom $\pi_j(\beta_j^l \mid \beta_{-j}^{(t)})$
   actualizar el bloque $\beta_j^{l(t)}$ dentro de $\beta^{(t)}$
  end for j
 end for l
 aceptar $\beta^{(t)}$ como muestra de la iteración t
end for t

A continuación se derivan las ecuaciones que definen la distribución condicional de los bloques de parámetros B_j.

Distribución condicional.

La distribución condicional de cada bloque de variables β_j de un nodo N_j depende de los nodos padres e hijos de ese nodo:

$$\pi_j(\beta_j \mid \beta_{-j}) = \pi_j(\beta_j \mid pa(N_j) \cup ch(N_j))$$

En los nodos *«a priori»* el bloque β_j no está definido ($\beta_j = 0$) por tanto sólo hay que caracterizar π_j para los nodos del nivel de observación y latentes.

Condicionando a las matrices de covarianzas, π_j está caracterizada por un sistema de la forma:

$$Y_{*}j = X_{*j} \cdot \beta_j + \eta_j \mid \eta_j \sim N(0, I) \qquad (25)$$

El sistema 25 se obtiene extendiendo el sistema local (ecuaciones 18 ó 19), con las ecuaciones de los nodos padres que involucren incógnitas en el nodo N_j condicionado.

Sea N_i Є $pa(N_j)$, si N_i es un nodo latente entonces el aporte al condicionamiento está contenido en las ecuaciones del sistema equivalente a 19:

$$L_i^{-1} \cdot X_i \cdot \beta_i = L_i^{-1} \cdot \alpha_i + \eta_i \mid \eta_i \sim N(0, I), \Sigma_i = L_i \cdot L_i^T \qquad (26)$$

Si N_i es un nodo con información *«a priori»* entonces el aporte al condicionamiento está contenido en las ecuaciones del sistema 20.

De forma genérica el aporte de un nodo *latente* o *«a priori»* se puede escribir como:

$$Y_i = L_i^{-1} \cdot \alpha_i + \eta_i \mid \eta_i \sim N(0, I), \Sigma_i = L_i \cdot L_i^T \qquad (27)$$

donde $Y_i = L_i^{-1} \cdot X_i \cdot \beta_i$ para el *nodo latente* y $Y_i = L_i^{-1} \cdot \mu_i$ para el *nodo «a priori»*

Dado que N_i es un padre de N_j, entonces algunas componentes de α_i son incógnitas en N_j y la selección de esas variables se puede hacer mediante la función $\Gamma_{ij}^a(.)$ que «es parte» de la arista e_{ij}. Se denota por:

$C_{ij}^a = \Gamma_{ij}^a(C_{ij})$

$\sim C_{ij}^a = \{1, \ldots, |A_i|\} - C_{ij}^a$

$\alpha_{ij}^a = \alpha_i [C_{ij}^a]$

$\sim\alpha_{ij}^a = \alpha_i [\sim C_{ij}^a]$

$L_{ij}^a = L_i^{-1} [.,C_{ij}^a]$

$\sim L_{ij}^a = L_i^{-1} [.,\sim C_{ij}^a]$

donde los vectores y matrices anteriores se interpretan como:

- α_{ij}^a = (…,α_{ik}, …) tal que α_{ik} es componente de β_j.
- $\sim\alpha_{ij}^a$ = (…,α_{ik}, …) tal que α_{ik} es componente de algún β_s de N_s Є ch(N_i).
- L_{ij}^a representa la submatriz de L_i^{-1} en la que se han seleccionado las columnas correspondientes a α_{ij}^a.
- $\sim L_{ij}^a$ representa la submatriz de L_i^{-1} en la que se han seleccionado las columnas correspondientes a $\sim\alpha_{ij}^a$.

El sistema 27 se puede reescribir como:

$Y_i = L_{ij}^a \cdot \alpha_{ij}^a + \sim L_{ij}^a \cdot \sim\alpha_{ij}^a + \eta_i \mid \eta_i \sim N(0, I), \Sigma_i = L_i \cdot L_i^T$

Si se denota $Y_{ij}^a = Y_i - \sim L_{ij}^a \cdot \sim\alpha_{ij}^a$, entonces:

$$Y_{ij}^a = L_{ij}^a \cdot \alpha_{ij}^a + \eta_i \mid \eta_i \sim N(0, I), \Sigma_i = L_i \cdot L_i^T \qquad (28)$$

El siguiente algoritmo describe los pasos a seguir para ensamblar el aporte de un nodo padre N_i a las ecuaciones que caracterizan la distribución condicional de un nodo hijo N_j.

```
Y: vector donde ensamblar el término independiente
   (Y*j de la ecuación 25)
X: matriz donde ensamblar la matriz de regresión
   (X*j de la ecuación 25)
base = Rows(X) : índice de la última ecuación ensamblada
ApplyCovarTransformation(i)
Obtain Cij, Cij^a, ~Cij^a , Lij^a, ~Lij^a, ~αij^a
Compute Yij^a = Yi - ~Lij^a · ~αij^a
for r = 1 to Rows(i)
 if Lij^a(r,.) is not null
  Y(base+r) = Yij^a(r)
```

```
    for c = 1 to |C_ij|
     X(base+r, Γ_ij^b(c)) = L_ij^a(r,c)
    end for c
   end if
  end for r
```

Distribución condicional observada.

La distribución condicional en un *nodo observado* N_j se obtiene planteando el sistema conjunto de la ecuación 18 y el aporte del sistema 28 según corresponda para cada uno de los nodos $N_i \in pa(N_j)$.

Las celdas de las matrices L_{ij}^a y los vectores Y_{ij}^a correspondientes a los padres del nodo se ensamblan dentro de la matriz $X_{*}j$ y el vector $Y_{*}j$ haciendo uso de la función `AssemblyEdge`. El algoritmo de construcción de la distribución condicional de un nodo observado se detalla a continuación.

$X = L_j^{-1} \cdot X_j$

$Y = L_j^{-1} \cdot Y_j$

for all $N_i \in pa(N_j)$

 AssemblyEdge(i,j,X,Y)

end for N_i

Distribución condicional latente.

La distribución condicional en un *nodo latente* N_j se obtiene de forma análoga al nodo observado partiendo del sistema local del nodo:

$$L_j^{-1} \cdot \alpha_j = L_j^{-1} \cdot X_j \cdot \beta_j + \eta_j \mid \eta_j \sim N(0, I)$$

y concatenando el aporte del sistema 28 según corresponda para cada uno de los nodos $N_i \in pa(N_j)$.

El correspondiente algoritmo de construcción de la distribución condicional se describe a continuación.

$X = L_j^{-1} \cdot X_j$

$Y = L_j^{-1} \cdot \alpha_j$

for all $N_i \in pa(N_j)$

 AssemblyEdge(i,j,X,Y)

end for N_i

Generador Gibbs paralelo.

Es posible explotar la independencia condicional entre grupos de nodos de un mismo nivel para paralelizar la iteración por los bloques de parámetros del *generador Gibbs*.

Si dos nodos N_i y N_j son independientes condicionalmente entonces estos pueden simularse en paralelo ya que la distribución condicional $\pi_j(\beta_j \mid \beta_{-j})$ no depende del vector de parámetros β_i y viceversa.

Los nodos de un nivel dado *l* pueden agruparse según esta propiedad de independencia condicional. Dos nodos N_i y N_j pertenecen a un mismo grupo G_k^l si N_i y N_j son hermanos con un padre común N_k y existe al menos una componente β_{ir} de α_{ki}^a y una componente βjs de α_{kj}^a con $E(\beta_{ir} \cdot \beta js) \neq 0$.

Los bloques de parámetros en nodos que son miembros de un mismo grupo deben simularse en orden uno detrás de otro debido a la correlación entre los parámetros. En cambio los grupos distintos pueden simularse en procesos independientes y paralelos debido a la independencia entre los grupos.

Seguidamente se describe la versión paralela del *generador Gibbs* a partir de una clasificación de los nodos en grupos independientes.

β vector de variables particionado en bloques
$(\beta_1^1, \ldots, \beta_{b1}^1, \ldots, \beta_1^L, \ldots, \beta_{bl}^L)$
$G = \{G_1^1, \ldots, G_{g1}^1, \ldots, G_1^L, \ldots, G_{gl}^L\}$ grupos según
independencia condicional por niveles
$G_j^l = \{\beta_{j1}^l, \ldots, \beta_{jk}^l\}$ conjunto de bloques de parámetros correlacionados
$\beta^{(0)} = (\beta_1^{1(0)}, \ldots, \beta_{b1}^{1(0)}, \ldots, \beta_1^{L(0)}, \ldots, \beta_{bl}^{L(0)})$ valor inicial del Gibbs
for t = 1 to M
  $\beta^{(t)} = \beta^{(t-1)}$
  for l = 1 to L
   for j = 1 to g_l (parallel)
    for all β_k in G_j^l
     $\beta_k^{l(t)}$ = SampleFrom $\pi_k(\beta_k^l \mid \beta_{-j}^{(t)})$
     actualizar el bloque $\beta_j^{l(t)}$ dentro de $\beta^{(t)}$
    end for β_k
   end for j (parallel)
  end for l
  aceptar $\beta^{(t)}$ como muestra de la iteración t
end for t

Todo esto se implementó en el propio lenguaje TOL por sus facilidades en el manejo de las estructuras necesarias para representar el problema, esto no debe dar motivo a

cuestionamiento alguno pues esta forma de proceder es muy común en el desarrollo de otros lenguajes de programación, donde las nuevas funciones a incluir son primeramente programadas y probadas en el propio lenguaje.

En la implementación de los algoritmos expuestos se deben tener en cuenta además algunos aspectos de la arquitectura informática.

Cuando el número de nodos no es excesivo o simplemente se cuenta con una sola máquina es conveniente contar con una arquitectura secuencial de estimación y manejo del modelo HLM, que además permite un acercamiento al problema más sencillo desde el punto de vista de la implementación.

Para los casos verdaderamente masivos es impensable un método secuencial pues podría tardar semanas o meses en resolver problemas que se deben ejecutar en apenas unas horas, por lo que es necesario contar con una arquitectura paralela, en la que varias máquinas trabajen en paralelo de la forma más escalar que sea posible, es decir, solapándose al máximo y trasmitiéndose el mínimo de información, para que el tiempo total empleado sea aproximadamente dividido por el número de máquinas.

En este sentido se implementan algunos elementos para soportar esta arquitectura que incluye:

- Un cluster de cálculo basado en TOL que facilita la paralelización de tareas de cálculo costosas como la simulación bayesiana.
- Un servidor que permite la ejecución y control de procesos en remoto.

Como resultado se obtienen tres herramientas:

- Un servidor de procesos remoto que se denomina RmtPS_Server.
- El cliente que no es más que un ejecutable TOL independiente denominado TOLSH.
- Un conjunto de funciones TOL que implementan la funcionalidad.

Más detalle de estas herramientas serán vistas en el capítulo siguiente.

Capítulo 3
"Discusión de los resultados"

En este capítulo se da una descripción de los factores de compresión obtenidos por el compactador de las tablas de llamadas telefónicas, se describen las funciones construidas para la clasificación vectorial, se explica el funcionamiento de las herramientas implementadas para el soporte de la arquitectura paralela necesaria en la estimación del modelo jerárquico lineal y por último se menciona otra de las funciones incorporadas a TOL.

3.1 Factores de compresión obtenidos.

La discusión se hace basado en la siguiente tabla, que no es más que un fragmento del reporte generado automáticamente por el compactador, teniendo como dato de entrada en el ejemplo concreto, la cantidad de 64,154,489 registros, que se corresponden con el número de llamadas de un mes.

--- bits by register ---

Field	value	quantity	msg	diction.	index	total	bytes
id_cliente	1.293	0.000	14.972	0.000	0.000	16.265	130432299
nrc	2.066	0.116	1.203	0.265	0.000	3.650	29273551
id_localidade_origem	1.216	0.050	1.129	0.000	0.000	2.395	19209416
numero_origem	2.469	0.052	2.220	0.004	0.000	4.745	38048522
id_localidade_destino	4.981	0.845	3.217	1.868	0.000	10.910	87491601
numero_destino	20.621	3.985	1.868	3.808	0.000	30.282	242839582
id_old_client	0.050	0.000	0.000	0.000	0.000	0.050	400800
id_subsegmento_cliente	0.609	0.061	0.004	0.000	0.000	0.674	5402460
id_carrier	0.243	0.000	0.973	0.000	0.000	1.216	9751523
id_degrau_tarifario	0.586	0.000	0.005	0.000	0.000	0.591	4740000
id_tipo_trafego	0.586	0.000	0.005	0.000	0.000	0.591	4740019
id_critica	0.586	0.000	0.005	0.000	0.000	0.591	4739982
id_tipo_chamada	0.586	0.000	0.005	0.000	0.000	0.591	4739982
data_hora_chamada	0.001	0.000	1.013	0.001	0.000	1.015	8137662
id_data_carga	0.002	0.000	1.372	0.001	0.000	1.375	11024926
segundos_duracao_real	0.328	0.000	8.931	0.002	0.000	9.261	74265329
other	0.000	0.000	4.580	0.000	0.000	4.580	36725966
total						88.781	711963620

Lo que se hace es simplemente describir, qué información es guardada en el fichero compactado que inciden en cada uno de los resultados mostrados en la tabla anterior.

Durante la lectura se debe siempre tener presente dos de los aspectos fundamentales que caracterizan a este compactador, en primer lugar su orientación al tiempo, es decir que los registros de la tabla que representan las llamadas efectuadas, son guardados en el fichero compactado ordenados temporalmente, o sea basados en el campo *data_hora_chamada*.

La otra característica es que el compactador aprovecha la naturaleza real del problema representado en la tabla a compactar, por lo que el análisis de los datos se hace por filas y no por columnas, es decir analizando las relaciones o dependencias entre los campos, teniendo como primer factor y más importante al cliente como fuente de decisión y a partir de aquí se hace un condicionamiento de unos campos respecto a otros.

En la implementación el condicionamiento entre los campos es el siguiente:

id_cliente ← nrc ← id_localidade_origem ← numero_origem

numero_origem ← id_localidade_destino ← numero_destino

numero_destino ← id_degrau_tarifario, id_tipo_trafego, id_critica, id_tipo_chamada

id_cliente ← id_old_client

nrc ← id_subsegmento_cliente, id_carrier

Como puede notarse, los campos *id_data_carga* y *segundos_duracao_real* no aparecen reflejados aquí, pues son campos temporales, es decir campos que a diferencia de los conceptuales, no dependen de un cliente individual, sino que su estructura depende del factor tiempo o sea del campo *data_hora_chamada*.

La descripción de los resultados obtenidos se hace por cada uno de los campos de la tabla, que se corresponden como puede apreciarse, con las filas de la tabla anterior a excepción del campo "*other*" que se explica al final.

Para lograr un mejor entendimiento, se explica primero de forma general el significado de cada columna de la tabla en cuestión, es decir, qué datos de los que son guardados en el fichero compactado por cada uno de los campos, inciden en las magnitudes reflejadas en estas columnas.

Columna *value.*

Esta columna se corresponde fundamentalmente con el espacio ocupado por los diferentes valores que toma cada campo en concreto, por ejemplo, para el campo *id_localidade_origem* en un caso específico, pudieran ser los valores 10820, 10864, 11272, etc., como debe suponerse, para el caso de los campos conceptuales, la

información que involucra esta columna está relacionada generalmente con las fichas de los clientes y en algunos campos con las listas de diccionarios globales.

Estos valores son almacenados usando diferentes estrategias de acuerdo con el campo en específico, pero de forma general para ahorrar espacio a la hora de almacenar una lista de valores, lo que realmente se almacenan son las diferencias de estos valores con el mínimo de entre todos los valores a almacenar, para lo cual sólo se necesitan [log2(max-min+1)]) bits para cada uno de ellos.

Si los valores se almacenaran ordenados por su magnitud, se pudieran usar estrategias más eficientes en el sentido del nivel de compresión, pero realmente en el compactador todas estas listas son guardadas en orden descendente de su frecuencia de aparición, esto se hace por dos razones fundamentales, en primer lugar precisamente para ahorrar espacio, pues al construir un diccionario Huffman para estos valores, entonces habría que guardar adicionalmente una asociación entre los índices usados por esta codificación y los índices al alfabeto original. El otro aspecto clave es para mejorar la localidad de referencia.

Columna *quantity.*

De forma general, cada vez que se va a almacenar una lista de valores según fue descrito en el punto anterior, se hace evidente la necesidad de conocer la cantidad de valores que conforman la misma, el espacio ocupado por estas cantidades son las que se reflejan en esta columna.

Para ilustrar esto de otra forma, haciendo referencia a los niveles de condicionamiento existente entre los campos que se describieron anteriormente, es aquí donde se especifica cuántos valores diferentes toma un campo determinado para un valor específico del campo en el nivel de anidamiento anterior.

Por supuesto que estas cantidades también son almacenadas siguiendo determinadas estrategias propias para cada campo, pero de forma general, estos valores son almacenados usando la cantidad de bits estrictamente necesarios para almacenar todas las cantidades referidas a un campo en específico.

Columna *diction*.

Dado que casi toda la información es compactada utilizando diccionarios Huffman, y debido a la gran cantidad de diccionarios que se hace necesario crear y por tanto almacenar, considerando la profundidad en el nivel de dependencia entre los campos, parece útil separar en esta columna el espacio que ocupa la información necesaria del

propio diccionario, es decir la información que es imprescindible almacenar para su persistencia o sea para poder reconstruirlos en el descompactador.

Para cada diccionario se almacena la longitud de palabra de código en bits mínima usada por este (campo *min_cw_len* en la componente *mrcoding*), la longitud de palabra de código en bits máxima (*max_cw_len*), y la lista de las frecuencias de cada una de las longitudes usadas (*cwlen_freq[]*), usando para esto la misma estrategia que se usó para almacenar una lista de valores en la columna *value*.

Columna *message.*

Esta columna es la que está más relacionada con la información a guardar para cada uno de los registros a compactar, aquí incide el tamaño ocupado por los mensajes de los registros, es decir las señales necesarias para saber a qué elementos de la ficha de un cliente se refiere cada registro, o en algunos campos a qué elemento de un determinado diccionario global se hace referencia.

Para el caso de los campos temporales, esta señal por supuesto no se refiere a elementos de ninguna ficha, aunque como es de suponer, también son codificadas usando algún diccionario Huffman y la estrategia seguida será explicada en cada campo.

Explicadas ya las columnas, se pasa ahora a hacer un análisis de cada uno de los campos describiendo los factores que intervienen en cada una de las columnas.

Es bueno notar de antemano que para todos los campos conceptuales, las columnas *value*, *quantity* y *diction.* se ven afectadas durante la construcción de las fichas de cada cliente, mientras la columna *message* se afecta durante la compactación de los registros.

id_cliente

En este campo sólo se ven afectadas dos columnas, *value* y *message*. Para el caso de la columna de los valores lo que se almacena aquí es precisamente la lista de todos los clientes que aparecen en el fichero de entrada tal y como se describió en el tópico sobre esta columna, aunque no es objetivo del informe hacer cálculos para mostrar cada uno de los valores que aparecen en la tabla, parece prudente a modo de ejemplo y por su sencillez detallar el valor que aparece en esta columna.

Este valor no es más que el resultado de dividir la cantidad de bits necesarios para almacenar esta lista entre el número total de registros de la tabla, el número de bits para almacenar la lista no es más que:

(# de valores de la lista) * (# de bits necesarios para almacenar cada uno)

En nuestro ejemplo aparecen 3,189,807 valores diferentes para este campo, todos en el rango [0,39000000), entonces la cantidad de bits necesarios para almacenar las

diferencias de estos con el valor mínimo (en nuestro caso 0) es 26 pues 2^{26} = 67,108,864, por lo tanto resulta que el valor que aparece en la tabla es igual a:

3,189,807 * 26 / 64,154,489 = 1.293

La otra columna a tener en cuenta aquí es la de los mensajes donde se observa que toma el considerable valor de 14.972, esto no es más que el resultado de almacenar para cada registro una referencia al cliente que efectuó dicha llamada, esto es hecho como es de esperar guardando el código Huffman correspondiente a dicho cliente.

Este valor, a pesar de ser uno de los de mayor peso en el total obtenido, no puede ser reducido pues es imprescindible, según la estructura del fichero compactado, que se almacene dicho valor en cada registro para poder saber qué ficha de cliente utilizar para decodificar los mensajes del registro específico.

nrc, id_localidade_origem, numero_origem

Estos campos se analizan en conjunto, pues los valores que toman en cada una de las columnas de la tabla, están muy relacionados con la interrelación que exista entre ellos de acuerdo a la clasificación del cliente, así por ejemplo si el cliente es de tipo (1,1,1) no aporta valor en las columnas *quantity*, *message* ni *diction*., mientras que si es del tipo (1,M,M) intervienen en estas columnas los campos *id_localidade_origem* y *numero_origem* mientras que el campo *nrc* no incide en ellas.

La columna *value* para cada uno de estos campos se analiza de forma similar al campo *id_cliente*, teniendo en cuenta que cuando se refiere a "# de valores de la lista", no es la cantidad de valores diferentes globalmente en todo el fichero de entrada, sino considerando la relación entre los campos, así para cada uno de los clientes aparece su lista de *nrc*, y para cada *nrc* de cada cliente aparece una lista de todos sus *id_localidade_origem* y así sucesivamente de acuerdo al condicionamiento de cada campo respecto al anterior según se vio al principio del informe.

De aquí se deduce que la cantidad de elementos a guardar en el fichero compactado para cada campo, vaya aumentando en la medida que se avanza en el nivel de anidamiento, en el ejemplo la cantidad de valores a almacenar es la siguiente:

nrc	3,898,479
id_localidade_origem	3,899,825
numero_origem	4,168,740

Para las columnas *quantity*, *diction*. y *message* la medida de la afectación está dada nuevamente por el tipo del cliente, es decir cuando un campo participa con valor 1 en la relación entre ellos no afecta estas columnas, mientras que cuando participa con valor M,

es necesario informar la cantidad de valores del campo en la columna *quantity*, crear un diccionario Huffman para sus valores afectando la columna *diction*. y especificar en cada uno de los registros en donde aparezca, una señal indicando a cuál de los elementos de la lista hace referencia, incidiendo en la columna *message*.

Es interesante notar que para los casos en que aparecen más de un campo participando con el valor M en la relación, se crearon estrategias para el tratamiento de estas multiplicidades, las mismas no son tan elementales por lo que no serán explicadas en este informe, lo que sí se puede decir es que contribuyen también a la disminución de los valores de estas columnas.

id_localidade_destino, numero_destino

Estos dos campos también se analizan en conjunto, pues según la estrategia seguida por el compactador ellos están relacionados en lo que se denomina la agenda del cliente. Es precisamente en estos dos campos, junto con el campo *id_cliente* que ya fue analizado, donde se obtienen los totales mayores en la tabla de los resultados de compresión, 10.910 y 30.282 bits por registro respectivamente.

Si se observa bien, se puede ver además como entre los campos conceptuales, es precisamente en éstos donde las columnas *value*, *quantity* y *diction*., que son las que intervienen en la confección de la ficha del cliente, alcanzan sus máximos valores, este resultado se debe al nivel en que se encuentran estos campos en la cadena de dependencias, o sea en los últimos niveles.

Refiriéndose a la columna *value* se observa como el número total de valores para estos campos que es necesario almacenar aumentó drásticamente:

id_localidade_destino	15,977,708
numero_destino	30,065,909

Aquí se ve como el número total de *id_localidade_destino* en el ejemplo alcanza casi la cuarta parte de los registros y el total de *numero_destino* casi la mitad de los registros, entonces al tener que representar tantos elementos la columna *value* alcanza los elevados valores 4.981 y 20.621 respectivamente.

Este elevado nivel de anidamiento afecta también a la columna *quantity*, al tener que guardarse tantos valores, o sea la cantidad de *id_localidade_destino* por cada *numero_origem* y la cantidad de *numero_destino* por cada *id_localidade_destino*, también aumenta la necesidad de confeccionar mayor número de diccionarios afectando la columna *diccion*.

Para disminuir los valores en *value* y por ende en *quantity* y *diccion*., se deben utilizar estrategias que aprovechen la estructura textual de estos campos.

id_old_client

En este campo los resultados obtenidos son satisfactorios por lo que las estrategias usadas son acertadas, sobre todo la de considerar como caso más general que este campo sea siempre cero para todos los registros del cliente, lo cual es indicado con un simple bit, sólo en caso contrario se necesita especificar la cantidad y los valores específicos que toma.

id_subsegmento_cliente

Los resultados obtenidos para este campo son favorables, por lo que las estrategias usadas son correctas, fundamentalmente la de considerar la estabilidad del subsegmento dentro del *nrc*, lo cual es indicado con un simple bit, sólo en caso contrario se necesita especificar la cantidad y los valores específicos.

id_carrier

Este campo está en el mismo nivel de profundidad que el campo anterior, pero se logró disminuir el valor de las columnas *value* y *quantity* aprovechando el hecho de que la diversidad de valores globales que toma es mucho menor, por tanto se usó la estrategia de utilizar una lista de diccionarios globales, de esta forma para cada *nrc* sólo es necesario guardar un índice a un diccionario dentro de esta lista, como la cantidad de diccionarios diferentes es pequeña, en el ejemplo sólo 15, bastó con 4 bits por cada *nrc* para representar este valor.

id_degrau_tarifario, id_tipo_trafego, id_critica, id_tipo_chamada

El tratamiento hecho a estos campos fue similar al anterior, incluyendo la lista de diccionarios globales, aunque con la diferencia de que se analizaron como cuartetas y no como campos independientes, para evitar la necesidad de tener que referenciar cuatro diccionarios.

data_hora_chamada

Se puede afirmar que el valor de 1.015 bits por registro obtenido en este campo, constituye el mejor resultado logrado por el compactador, aquí la propuesta fue seguida a cabalidad, este excelente resultado es también muy importante al ser el campo rector de toda la información en el compactador.

id_data_carga

Este es uno de los campos temporales, o sea que está subordinado directamente al campo *data_hora_chamada*, se siguió la estrategia planteada en la propuesta y los resultados obtenidos fueron excelentes, apenas 1.375 bits por registro.

segundos_duracao_real

Al ser este el otro campo temporal, se utiliza una estrategia bastante similar a la de *id_data_carga*, pero como los valores de este campo son bastante heterogéneos en el tiempo, el resultado obtenido es más alto.

other

Este campo con un valor resultante de 4.580, no es un campo propio de la tabla, sino que es un campo creado para almacenar la longitud de los mensajes de cada registro, por supuesto que lo que se escribe aquí es el código Huffman de estas longitudes, este campo es necesario para poder saltarse registros durante el proceso de descompactación sin necesidad de procesarlos, por lo que su valor debe permanecer sin cambios.

3.2 Funciones de clasificación.

Según se mencionó en el capítulo anterior, como etapa inicial en la confección del modelo jerárquico se encuentra la clasificación. Las funciones incorporadas a TOL para este propósito son las siguientes:

```
Cluster(Matrix data, Real k, Real n
        [, Text method="a", Text dist, Matrix weigth]);
```

La cual divide las filas de la matriz 'data' in 'k' grupos (clusters) de manera que la suma de las distancias de las filas al centro de su grupo (centriod) sea mínima, para lograr esto se basa en la aplicación del algoritmo EM (Expectation-Maximization) 'n' veces con el objetivo de encontrar la solución optima [HOO06].

Aquí se ofrecen varios métodos para calcular el centro de los grupos que pueden ser: la media aritmética, la mediana, o el elemento con menor distancia a todos los de su grupo (medoid).

```
HierarchicalCluster(Matrix data, Real k
              [, Text method="s", Text dist="e", Matrix weigth]);
```

A diferencia de la anterior basa la obtención de los grupos en un método totalmente determinista y no necesita por tanto de repeticiones para obtener la solución óptima, para ello construye un árbol a partir de las semejanzas entre los diferentes elementos a agrupar de acuerdo con la matriz de distancias entre todos ellos, de aquí su nombre de jerárquico.

En ésta se puede especificar el método usado para calcular la distancia entre los nodos del árbol basado en sus elementos, así se tiene que puede ser: la más corta de todas las distancias entre los pares de miembros de los dos nodos, la mayor de las distancias, el promedio de todas, o la distancia entre sus centros.

Ambas funciones tienen la ventaja de aceptar valores desconocidos (?) en sus filas, también se puede elegir entre diferentes funciones para medir la similitud o distancia entre los elementos como la clásica euclidiana, la "city-block", otras basadas en coeficientes de correlación de Pearson, etc., se puede además dar mayor peso en la selección a determinadas columnas.

Las funciones retornan un conjunto cuyos elementos son entre otros:

`Real AverageDistortion` - El promedio de las distancias de los elementos a su centro, puede utilizarse como medida del error de la solución encontrada.

`Real TimesSolutionFound` - La cantidad de veces que se encontró la misma solución óptima ofrecida, esto da una medida de la posibilidad de encontrar mejores soluciones ejecutando mayor cantidad de veces el algoritmo EM, por supuesto que esto sólo tiene sentido para la función Cluster().

`Matrix Assignments` - este arreglo almacena el número del grupo al cual fue asignado cada elemento.

`Matrix Centroids` - almacena los centros de los grupos.

`Set SetOfClusters` - cada subconjunto de este conjunto representa un grupo con sus elementos.

En ambas funciones los parámetros opcionales toman por omisión el valor para el cual la implementación se base en métodos más rápidos y que usen la memoria de forma más eficiente.

3.3 Herramientas para cálculos en paralelo.

Como se vio en el capítulo anterior, para la implementación de una arquitectura paralela se obtienen como resultado las siguientes herramientas:

RmtPS_Server

Es una aplicación servidor que permanece en ejecución por un largo período de tiempo y su funcionalidad consiste en atender peticiones remotas vía "socket".

Sus funciones están relacionadas con el control de procesos, o sea, arrancar un programa, verificar un proceso y terminar un proceso. La comunicación entre el cliente y el servidor es orientada a línea.

RmtPS_Server registra los eventos y las peticiones atendidas tanto exitosas como fallidas de forma tal que se facilita la labor de diagnosis por parte de los usuarios y administradores.

En el caso de plataforma Windows se logra que se comporte como un servicio Windows, este será manejado por winserv.exe, para obtenerlo se deben seguir las instrucciones en http://wiki.tcl.tk/12798.

Para instalar el servicio basta con ejecutar desde una ventana del DOS la siguiente línea

```
winserv install RMTPS_W_SERVER -start auto rmtps_server.exe
```

donde RMTPS_W_SERVER es el nombre del servicio en Windows y para desinstalarlo se debe detener el servicio y ejecutar desde una ventana del DOS

```
winserv uninstall RMTPS_W_SERVER
```

Los mensajes generados por el servicio (logs) pueden verse en el visor de sucesos de Windows.

TOLSH

Es un nuevo ejecutable TOL en modo de línea de comandos, capaz de llamar a Tcl basado en la extensión TOLTcl que permite comunicar Tcl con TOL.

Este ejecutable permite ejecutar sentencias y ficheros TOL directamente desde la línea de comandos del sistema operativo, sin tener que cargar una interfaz de usuario.

Con la opción -c se permite pasar código TOL directamente en la línea de comandos de la siguiente manera:

```
TOLSH –c "Real a = 1; Real b = 2; Real c = a + b;"
```

El comando ejecuta las líneas de código TOL entrecomilladas y termina. Aparentemente parece que no ha ocurrido nada, pues no se muestra ningún tipo de salida, sin embargo las sentencias han sido ejecutadas.

Si se tiene interés en obtener más información al ejecutar las sentencias, se puede utilizar la opción `–c` acompañada de la opción `–v` y se obtiene:

```
Cargando la Tol Init Library /usr/local/lib/tolInitLib/_inittol.tol
1.000000
2.000000
3.000000
```

Como se vio, se puede introducir más de una sentencia Tol entre comillas, el resultado de la ejecución de cada sentencia se devuelve a la salida estándar o a un fichero log especificado en la configuración del cliente.

Funciones TOL

Las funciones que la parte cliente expone son las siguientes:

```
Real RemotePing(Text Host, Real Port);
```

Verifica si un servidor RMTPS_SERVER está ejecutándose en un ordenador remoto y está escuchando peticiones.

```
Real RemoteExec(Text Host, Real Port, Text CmdLine);
```

Ejecuta en un ordenador remoto un comando dado en *CmdLine*, este consiste en el programa a ejecutar y sus argumentos.

Retorna el PID del proceso ejecutado o 0 si algún error ocurrió durante el intento de ejecución del comando.

```
Real RemoteAlive(Text Host, Real Port, Real PID);
```

Verifica si un proceso dado por su PID continúa vivo en un ordenador remoto.

Retorna 1 si el proceso está en la lista de procesos activos, 0 si el proceso no está activo o -1 si hubo un error durante la operación remota, por ejemplo: PID inválido.

```
Real RemoteKill(Text Host, Real Port, Real PID);
```

Termina la ejecución de un proceso, dado por su PID, en un ordenador remoto.

Estas tres funciones necesitan que un servidor RmtPS_Server esté escuchando en el puerto Port del ordenador Host.

Se necesita que la aplicación sea multiplataforma, es decir soportable sobre Windows y Linux, es por ello que se decidió implementarla en Tcl, este paquete es muy común entre todas las distribuciones Unix, además hay bastante código fuente escrito en Tcl disponible en Internet, y existe ya una extensión TOLTcl que permite comunicar Tcl con TOL que facilita mucho la implementación de TOLSH, además de poder tener toda la potencia de Tcl dentro de TOL.

En esencia, la comunicación entre el servidor y el cliente se desarrolla de la siguiente manera:

1) El servidor atiende peticiones por el puerto configurado a tales efectos.
2) El cliente envía una línea con una petición completa, establece una escucha en espera de la respuesta del servidor.
3) El servidor recibe la petición, la despacha inmediatamente, deja la respuesta de la petición en el canal del cliente y continúa atendiendo peticiones.

4) La escucha en el cliente se activa y la respuesta es leída inmediatamente desde el canal del servidor.

Como ejemplo ilustrativo se muestra el código Tcl de la implementación del núcleo del cliente:

```
proc ::rmtps_client::ask_server { host port request } {
  variable server_result
  variable server_timeout
  variable timeout_id
  # Open channel to server
  set chan [socket $host $port]
  # We want whole lines
  fconfigure $chan -blocking 0 -buffering line
  # Send the request to the server
  puts $chan $request
  # Remember that server is line oriented
  flush $chan
  # Set up a fileevent to handle input from server
  fileevent $chan readable [namespace code "wait_result $chan"]
  # Set up timeout handler
  set timeout_id [after [expr {1000*$server_timeout}] \
                      [namespace code "handle_timeout $chan"]]
  # Wait for result or timeout
  vwait ::rmtps_client::server_result
  catch {close $chan}
  set server_result
}
```

3.4 Otras funciones incorporadas a TOL.

Por último se debe mencionar que son muchas las funciones que fueron mejoradas o incluso otras incorporadas a TOL para poder desarrollar los algoritmos que dan solución a la estimación del modelo presentado, por la razón de no hacer tan extenso el documento no se detallan, sólo a modo ilustrativo se hace referencia a una de ellas por su relación directa con el modelo jerárquico lineal.

Se trata de la implementación de la función:

```
Matrix InnerPoint(Matrix B, Matrix c);
```

la cual encuentra un vector x de R^n, tal que Bx < c, donde B es una matriz de dimensión mxn y c el vector de términos independientes de dimensión mx1.

La necesidad de implementar la misma surge recientemente, motivado por lo que expondremos a continuación.

En la modelación masiva como la que se ha tratado es bastante habitual que el número de variables crezca desmesuradamente dada la complejidad de los sistemas de promoción, efectos calendario, meteorológicos, etc., que se entrelazan entre sí en formas que no siempre son fáciles de parametrizar escuetamente. Al aumentar los grados de libertad, los modelos estimados por máxima verosimilitud de forma independiente pueden sobreajustar los datos.

Uno de los objetivos fundamentales de la estimación bayesiana de un Modelo Jerárquico Lineal, es restringir de forma probabilística el conjunto de valores que pueden adoptar los parámetros que intervienen en los distintos modelos.

En el caso del HLM normal con restricciones de desigualdad, las relaciones entre un conjunto de variables aleatorias se deberán expresar paramétricamente diciendo que dichas variables, o cierta combinación lineal de ellas, tienen una distribución conjunta multinormal truncada por un conjunto de restricciones de desigualdad lineal [ROD05].

La implementación del modelo HLM permite la formulación de distintos tipos de restricciones, dentro de éstas se tienen las restricciones de desigualdad, que son las que acotan una combinación lineal de parámetros en una región convexa mediante desigualdades lineales expresables mediante un nodo de dominio.

En los nodos de dominio no se introducen nuevas variables sino que se postulan restricciones de desigualdad lineal entre dichas variables

$$A\beta \leq a$$

Precisamente como un paso intermedio en la implementación del modelo se necesita encontrar un punto interior a esta región convexa acotada por las restricciones, de aquí el nombre dado a la función, indicando que la misma encuentra un punto interior a la región o poliedro que representan las soluciones del sistema [CAS02].

La implementación de la función se basa en una consecuencia inmediata de la definición de cono dual, ésta plantea que resolver el sistema de ecuaciones e inecuaciones $Ax \leq 0$ es equivalente a obtener los generadores del cono dual del cono generado por las filas de la matriz A.

Para obtener un conjunto minimal de generadores de un cono dual se implementa el algoritmo Γ [CAS04].

Los conos duales son muy útiles para obtener conjuntos de generadores minimales del espacio vectorial, el cono y el politopo que definen el poliedro de las soluciones factibles de un sistema de la forma $Bx \leq c$, que es el que se necesita encontrar.

Con este objetivo, se reemplaza el espacio euclidiano R^n por otro espacio euclidiano R^{n+1}, usando este truco, se convierten los poliedros en R^n en conos en R^{n+1}, por ello, sólo tiene que tratarse con conos, sin embargo, al final, se necesita volver al espacio euclidiano de partida R^n.

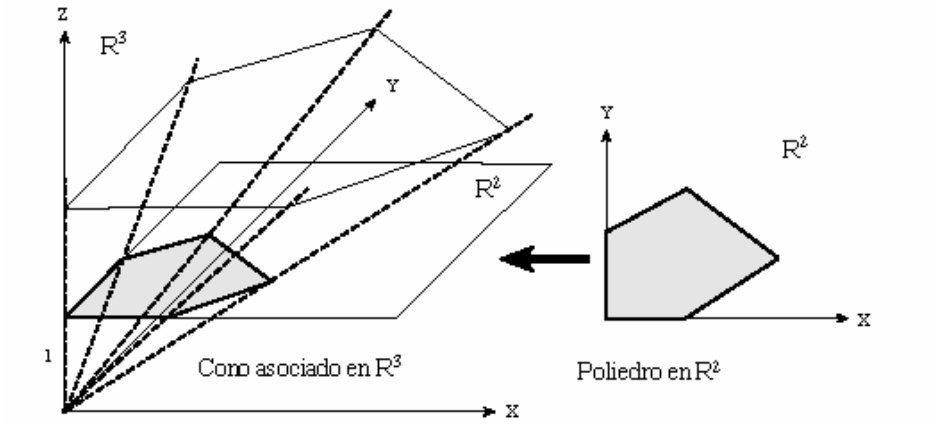
Más precisamente, el poliedro

$$S = \{x \in R^n \mid Hx \leq a\}$$

se reemplaza por el cono

$$C_S = \{(X\ x_{n+1})^T \in R^{n+1} \mid Hx - x_{n+1}a \leq 0;\ -x_{n+1} \leq 0\}$$

Este proceso se ilustra en la siguiente figura, donde se comienza con el pentágono (poliedro) en R^2 y se asciende a R^3 añadiendo una componente más a sus vértices (puntos extremos).



Seguidamente, se considera el cono generado por estas direcciones extremas, una vez que se está en R^{n+1}, se buscan los generadores del cono dual, usando el algoritmo Γ.

Finalmente, se recuperan las restricciones iniciales en R^n imponiendo la restricción $x_{n+1} = 1$, es decir, obteniendo la intersección del cono con el hiperplano $x_{n+1} = 1$.

Finalmente, para obtener una solución en particular basta con hacer una combinación lineal convexa de estos vectores generadores.

Conclusiones

Con este trabajo se logra enriquecer el lenguaje de programación TOL, al incorporarle un conjunto de funciones y herramientas que lo convierten en un instrumento más potente para la modelación estadística, dentro de los aportes se tiene:

- Con la propuesta de compactación implementada, se logran excelentes factores de compresión, buena velocidad de descompresión y acceso aleatorio a los registros de las tablas de manera eficiente, permitiendo todo esto un efectivo análisis de los datos del cliente para poder ofrecer la respuesta que esperan.
- Durante la implementación se asimilaron y utilizaron un conjunto de herramientas novedosas de programación existentes para el manejo de estructuras de datos y algoritmos en memoria externa que se pueden usar para desarrollar otras aplicaciones.
- Como un producto del trabajo se derivaron un conjunto de bibliotecas para el manejo eficiente de grandes ficheros, para el acceso a nivel de “bit” y para la compactación, que se pueden utilizar en otros trabajos.
- Se logró la implementación de un modelo jerárquico lineal bayesiano, del cual no se tiene referencia de su existencia en el área de la modelación estadística, con la metodología propuesta y orientado a manipular grandes volúmenes de información.
- Con la introducción de esquemas jerárquicos bayesianos en la modelación expuestos en este documento, se mejoran los actuales mecanismos de estimación para poder lograr una mejora del sistema de distribución.
- El lenguaje se enriqueció con la incorporación de un conjunto de nuevas funciones dentro de las que se destacan las referidas a la clasificación vectorial y a la resolución de sistemas de inecuaciones lineales.
- Se construyeron herramientas que soportan la arquitectura informática necesaria en la resolución de modelos masivos a través de la ejecución de procesos de cálculo costoso de forma paralela.

Recomendaciones

- Implementar un lenguaje de definición que permita especificar las características de los campos de las tablas a compactar, para aplicar las técnicas ya programadas a otras tablas con estructura similar o que se adapten a los métodos explicados.
- Implementar una versión de los algoritmos del modelo jerárquico lineal en C++ con el objetivo de aumentar la velocidad de ejecución, quedando así además construido dentro del lenguaje TOL y no como una extensión de este.
- Proponer e implementar métodos relativos a la compactación de la ficha que utilicen las características textuales de los campos, con el objetivo de lograr mayores factores de compresión.
- Implementar e incorporar otros métodos de clasificación como Self-Organizing Maps, y Principal Component Analysis.

Bibliografía

[AHO72] Alfred V. Aho, Jeffrey D. Ullman, “The Theory of Parsing, Translation and Compiling”, Prentice-Hall, 1972.

[ARE01] Arellano, M., "Introducción al Análisis Clásico de Series de Tiempo". http://www.5campus.com/leccion/seriest. 2001.

[ART04] Luis M. Artiles Martínez, “Desarrollo práctico de la propuesta - Compactación de llamadas e IDDW”, Documento Técnico, Bayes Forecast, Mayo 2004.

[ART07] Luis M. Artiles Martínez. “Estimación masiva de parámetros para modelos lineales jerárquicos”. Documento Técnico, Bayes Forecast, 2007.

[BOL94] Bolotin, V. A., Modeling Call Holding Time Distributions for CCS Network Design and Performance Analysis. IEEE Journal on Selected Areas in Communications, 12(3):433--438, April 1994. (pp 33, 164, 166).

[BOL96] Bolotin, V. A. Modeling Call Holding Time Distributions for CCS Network Design and Performance Analysis. IEEE Journal on Selected Areas in Communications, 14(4), 1996.

[BOX94] George Box, Gwilym M. Jenkins and Gregory, “Time series analysis: Forecasting and control”, Prentice-Hall, 2nd edition, 1994.

[BUE02] Víctor de Buen Remiro, “Compactador de tablas de llamadas telefónicas HBF”, Documento Técnico, Bayes Forecast, Marzo 2002.

[CAS92] George Casella and Edward I. George. “Explaining the gibbs sampler”. The American Statistician, 46(3):167–174, Agosto 1992.

[CAS02] E. Castillo, F. Jubete, E. Pruneda and C. Solares. “Obtaining simultaneous solutions of linear subsystems of equations and inequalities”, Linear Algebra and its Applications, 346, 131-154, 2002.

[CAS04] E. Castillo and F. Jubete. “The Γ-algorithm and Some Applications”, International Journal of Mathematical Education in Science and Technology, 35, 3, 369-389, 2004.

[CHA75] Chao, Lincoln L., “Estadística para ciencias sociales y administrativas”. Bogotá, McGraw-Hill. 1975.

[CHA96] Gary Chartrand and L. Lesniak. “Graphs & Digraphs”. Chapman Hall/CRC, 3 edition, August 1996.

[CHE00] Chen, C.W.S., Lee, J.C., Lee H.Y, Niu, W.F., “Bayesian Estimation for Time Series Regressions improved with exact likelihood”, Journal of Statistical Computation with Statistics, 2000.

[CHI95] Siddhartha Chib and Edward Greenberg. “Understanding the metropolis-hastings algorithm”. The American Statistician, 49(4):327–335, November 1995.

[GEL03] Andrew Gelman, John B. Carlin, Hal S. Stern and Donald B. Rubin, “Bayesian Data Analysis”, CRC Press, 2nd edition, 2003.

[GOH02], Goharian N, Ankit Jain, Qian Sun, “Comparative Analysis of Sparse Matrix Algorithm for Information Retrieval”, 2002.

[HAM94] James D. Hamilton, “Time Series Analysis”, Princeton University Press, 1994.

[HOO06] Michiel de Hoon, Seiya Imoto, Satoru Miyano; “The C clustering library for cDNA microarray data”, The University of Tokyo, Institute of Medical Science, Human Genome Center, June 2006.

[KAA98] Kaas, R.E., Carlin, B.P., Gelman, A., Neal, R.M: “Markov chain Monte Carlo in Practice: A roundtable discussion”, American Statistical Asociation, Vol. 52-2, 1998.

[KNU73] Donald E. Knuth, “The Art of Computer Programming”, Volume 3, “Sorting and Searching”, Addison-Wesley Publishing Company, 1973.

[KOR99] Marcel Kornacker, “High-Performance Extensible Indexing”, Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

[MAK83] Makridakis, S; Wheelright, S.C.; McGee, V.E., “Forecasting: Methods and Applications”. Wiley, New York. 1983.

[MOF94] Moffat, A.; Sharman, N.; Zobel, J., “Static compression for dynamic texts”, Data Compression Conference, 1994. DCC '94. Proceedings, 29-31 March 1994 Pages: 126-135.

[MOF97] Moffat, A.; Turpin, A., “On the implementación of Minimum Redundancy Prefix Codes”; IEEE Transactions on Communications, Volume: 45, Issue: 10, Oct. 1997 Pages:1200 - 1207.

[MOF01] Moffat, A.; Turpin, A., “Efficient construction of minimum-redundancy codes for large alphabets”; Information Theory, IEEE, April 2001.

[MOF02] Alistar Moffat & Andrew Turpin, “Compression and Coding Algorithms”; Kluwer Academic Publishers, February 2002.

[PEÑ89] Peña, Daniel., “Estadística, Modelos y Métodos 2”. ”Modelos Lineales y Series Temporales”. Alianza Universidad, Madrid. 1989.

[PER05] Jorge S. Pérez Ronda, “Conjunto de herramientas para la codificación y decodificación”, Documento Técnico, Bayes Forecast, Noviembre 2005.

[PER06] César Pérez Alvarez. “Estimación jerárquica de los modelos de tráfico”, Documento Técnico, Bayes Forecast, Enero 2006.

[PER07] Jorge S. Pérez Ronda, “Descripción del Grafo HLM”, Documento Técnico, Bayes Forecast, Marzo 2007.

[ROB05] Christian P. Robert and George Casella. “Monte Carlo Statistical Methods”. Springer-Verlag, 2005.

[ROD05] Gabriel Rodríguez -Yam, Richard A. Davis, and Louis L. Scharf, “Eficient Gibbs Sampling of Truncated Multivariate Normal with Application to Constrained Linear Regresión”, 2005.

[RUS05] Daniel Rus Morales, “Lenguaje de Programación TOL”, Documento Técnico, Bayes Forecast, Junio 2005.

[SAL00] Salomon, D., “Data Compression” The complete Reference, 2nd ed., 2000

[SAY00] Sayood, K. “Introduction to Data Compression”, 2nd Ed., Academic Press, 2000.

[TUR00] Turpin, A.; Moffat, A., “Housekeeping for Prefix Coding”; IEEE Transactions on Communications, Volume: 48, Issue: 4, April 2000 Pages: 622-628.

[WIK07a] The Free encyclopedia Wikipedia. “Bayesian network”. http://en.wikipedia.org/wiki/Bayesian_network, Mayo 2007.

[WIK07b] The Free encyclopedia Wikipedia. “Graph (mathematics)”. http://en.wikipedia.org/wiki/Graph_(mathematics), Mayo 2007.

[WIT94] Witten I.H., Moffat A., Bell T., “Managing Gigabytes: Compressing and Indexing, Documents and Images”, 1st ed., 1994.