



Universidad
de La Laguna

Universidad de La Laguna
Escuela Superior de Ingeniería y Tecnología
Sección de Ingeniería Informática

INGENIERÍA EN INFORMÁTICA
PROYECTO FINAL DE CARRERA

DisPar: a parallel code to simulate
observational effects,
and
MkBundle+: a graphical interface to create
the input file of a code to calculate star
formation histories

Alumnos:

Anthony Vittorio Russo Cabrera
Manuel Luis Aznar

Tutores:

Juan José Salazar González
Inmaculada Rodríguez Martín

San Cristóbal de La Laguna, 10 de junio de 2016

Inmaculada Rodríguez Martín, con NIF 43360612T, y **Juan José Salazar González**, con NIF 43356435D, profesores titulares de la Universidad de La Laguna adscritos al departamento de *Matemáticas, Estadística e Investigación Operativa* en el área de *Estadística e Investigación Operativa*, como tutores:

CERTIFICAN

Que la presente memoria titulada:

***DisPar*: a parallel code to simulate observational effects,
and
MkBundle+: a graphical interface to create the input file of a code to
calculate star formation histories**

ha sido realizada bajo su dirección por **Anthony Vittorio Russo Cabrera**, con NIF 79063141C, y **Manuel Luiz Aznar**, con NIF 54057961A.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en San Cristóbal de La Laguna, a 10 de junio de 2016.

Agradecimientos

A Eva y a mi madre por los sacrificios
realizados y por el apoyo continuo.

A mi compañero por su constancia
y por su dedicación.

A los tutores por sus ánimos.

Anthony Vittorio Russo Cabrera
<anthony.russo.cabrera@gmail.com>

A mis padres y hermano
Aurora, Domingo y Enrique.

A Lucy y Javito.

A mi compañero por su
incansable dedicación.

A todos mis compañer@s
de carrera.

A los tutores por todos
los apoyos recibidos.

Manuel Luis Aznar
<manuel.luis.aznar@gmail.com>

Abstract

The main objective of this project is to develop two applications for *'The Local Group in Multi-Dimensions'*, at the Instituto Astrofísico de Canarias (IAC).

The first application is a Command Line Interface (CLI) for GNU/Linux systems. The group already has this application; however, the computation time that it takes to solve the problem is somewhat high. We have developed two applications to provide different functionalities, both of them reduce the computation time.

The second application is a multiplatform Graphical User Interface (GUI). This application will use the data produced by the first applications (or by different sources) showing it on the screen in a friendly way to be easily understood by the user. The user could interact with the application through mouse and keyboard events to define new information. This application reduces the time invested by the user to define new information.

Resumen

Este proyecto tiene como objetivo desarrollar dos aplicaciones para el *Grupo de Investigación 'Evolución de galaxias en el Grupo Local'*, del IAC.

La primera aplicación es una interfaz de línea de comandos (CLI) destinada a sistemas GNU/Linux. El grupo de investigación dispone de esta aplicación, sin embargo, el tiempo de computo consumido por la misma es algo elevado. Se han desarrollado dos aplicaciones que proporcionan distintas funcionalidades y reducen el tiempo de computo.

La segunda aplicación es una interfaz gráfica de usuario (GUI) multiplataforma. Esta aplicación usará la información producida por la primera aplicación (o por otras fuentes) mostrándola en pantalla de forma amigable para que el usuario la pueda interpretar fácilmente. El usuario podrá interactuar con la aplicación mediante eventos de ratón y teclado para definir nueva información. La aplicación desarrollada en esta segunda parte reduce en gran medida el tiempo que debe invertir el usuario para definir nueva información.

Índice general

Prefacio	XIII
I Mejora de rendimiento	1
1 Introducción	3
1.1 Descripción general del problema y consideraciones	3
1.2 Descripción de los elementos del problema	3
2 Descripción del problema y parámetros de entrada	5
2.1 Especificación de los parámetros de entrada	5
2.2 Pseudocódigo	5
3 Cuestiones previas a la implementación	9
3.1 Descripción y configuración utilizada para la ejecución de pruebas	9
3.2 Validación de los resultados obtenidos en las pruebas	11
4 Implementación y tiempos obtenidos: Pseudocódigo	13
5 Mejora del rendimiento: Estructura Índice AVL	19
6 Paralelismo a nivel de CPU y a nivel de GPU: OpenMP y CUDA	25
6.1 OpenMP: Paralelismo a nivel de CPU	25
6.2 CUDA: Paralelismo a nivel de GPU Nvidia	26
7 Implementación recursiva con paralelismo sobre CPU	29
7.1 Introducción	29
7.2 Herramientas de desarrollo	30
7.2.1 Requisitos	30
7.2.1.1 Compilador	30
7.2.1.2 Librerías y herramientas de terceros	30
7.2.1.3 Entorno de desarrollo	31
7.2.2 Instalación de herramientas	31
7.2.3 Configuración de herramientas	32
7.2.3.1 Generar librerías estáticas	32
7.2.3.2 Objetivos de compilación	32

7.3	Implementación de la aplicación	34
7.3.1	Árbol de directorios	35
7.3.2	Estructura de datos	36
7.3.3	Diagrama de clases	37
7.3.3.1	ArtificialStar	38
7.3.3.2	RealStar	38
7.3.3.3	DispersedStar	39
7.3.3.4	Epsilon	39
7.3.3.5	Node	39
7.3.3.6	Tree	41
7.3.3.7	Element	42
7.3.3.8	List	42
7.3.3.9	Index	43
7.3.4	Funciones destacadas	44
7.3.5	Recursividad	45
7.3.6	Analizadores	45
7.3.7	Documentación	45
7.4	Manual de usuario	46
7.4.1	Funcionamiento	46
7.4.2	Parámetros	47
7.4.3	Compilación	48
7.4.4	Ejemplo	48
7.4.4.1	Parámetros	48
7.4.4.2	Fichero de estrellas artificiales	49
7.4.4.3	Fichero de estrellas reales	49
7.4.4.4	Estructuras de datos	49
7.4.4.5	Búsqueda	50
7.4.4.6	Dispersión	50
7.4.4.7	Resultados	50
7.5	Resultados	50
7.6	Ganancia de velocidad	52
8	Implementación: Paralelismo a nivel de CPU y de GPU	55
8.1	Introducción	55
8.2	Requisitos: Hardware y software. Instalación del software	56
8.3	Formato de entrada: Ficheros de Parámetros	57
8.4	Almacenamiento de datos	59
8.5	Lectura de ficheros de datos	61
8.6	Cálculo de expresiones. Características madre y crowding	62
8.7	Creación de la estructura del capítulo 5: Índice AVL	64
8.8	Linealización de la estructura Índice AVL: Índice Vector	66
8.9	Procesado de estrellas madre: Dispersión	69

8.10	Escritura de resultados en disco.	72
8.11	Modo de uso: Compilación y ejecución	73
8.12	Prueba de rendimiento	74
8.13	Conclusiones	79
9	Comparación de las implementaciones	81
II	Interfaz gráfica	85
10	Aplicación de escritorio con entorno gráfico	87
10.1	Marco teórico	87
10.2	El problema	88
10.2.1	Descripción	88
10.2.2	Prototipado de la aplicación	89
10.3	Herramientas utilizadas	91
10.3.1	Selección	91
10.3.2	Requisitos	91
10.3.3	Sugerencias	91
10.3.4	Instalación	92
10.3.5	Configuración	93
10.3.6	Distribuciones	93
10.4	Descripción de la aplicación	94
10.4.1	Árbol de directorios	94
10.4.2	Descripción de clases	95
10.4.2.1	MainWindow	95
10.4.2.2	ManageDiagram	97
10.4.2.3	Diagram	98
10.4.2.4	Bundle	99
10.4.2.5	Histogram	100
10.4.2.6	NavigationToolbar	101
10.4.3	Eventos de la aplicación	101
10.4.4	Formato de los ficheros	102
10.4.4.1	Ficheros de parámetros	102
10.4.4.2	Ficheros de datos	104
10.5	Manual de usuario	104
10.5.1	Modos de ejecución	105
10.5.1.1	Usando el código fuente	105
10.5.1.2	Elijiendo algunas de las distribuciones ofrecidas	105
10.5.2	Visión general	105
10.5.2.1	Acciones principales	106
10.5.2.1.1	Open settings	106
10.5.2.1.2	Save settings	106

10.5.2.1.3	Open parameters	106
10.5.2.1.4	Save parameters	106
10.5.2.1.5	Open observed diagram	107
10.5.2.1.6	Open model diagram	107
10.5.2.1.7	Show/Hide diagram tool	107
10.5.2.1.8	Show/Hide bundle tool	107
10.5.2.1.9	Show/Hide histogram tool	108
10.5.2.1.10	Reset original view	108
10.5.2.1.11	Back to previous view	108
10.5.2.1.12	Forward to next view	108
10.5.2.1.13	Pan axes with left mouse, zoom with right	108
10.5.2.1.14	Zoom to rectangle	109
10.5.2.1.15	Select a point	109
10.5.2.1.16	Save the figure	109
10.5.2.1.17	Create bundle	109
10.5.2.1.18	Open bundle	109
10.5.2.1.19	Close bundle	110
10.5.2.1.20	Remove bundle	110
10.5.2.1.21	Create point	110
10.5.2.1.22	Modify point	110
10.5.2.1.23	Remove point	111
10.5.2.1.24	About this	111
10.5.2.1.25	About Qt	111
10.5.2.1.26	Exit application	111
10.5.2.2	Disposición general	112
10.5.2.2.1	Menu Bar	112
10.5.2.2.2	Status Bar	113
10.5.2.2.3	Toolbar	113
10.5.2.2.4	Ventana central	114
10.5.2.2.5	Ventanas flotantes	117
10.5.3	Casos de uso	120
10.5.3.1	Carga de datos	120
10.5.3.2	Creación de un bundle	121
10.5.3.3	Añadir punto a un bundle	122
10.5.3.3.1	Botón izquierdo del ratón	123
10.5.3.3.2	Botón derecho del ratón	124
10.5.3.3.3	Botón izquierdo del ratón sobre una recta	125
10.5.3.3.4	Botón izquierdo del ratón sobre un punto existente	127
10.5.3.4	Modificar un punto de un bundle	128
10.5.3.5	Eliminar punto de un bundle	129
10.5.3.6	Abrir un bundle	130

10.5.3.7	Eliminar un bundle	131
10.5.3.8	Cerrar un bundle	132
10.5.3.9	Establecer la rejilla de un bundle	133
10.5.3.10	Modificar el aspecto de los diagramas	134
10.5.3.11	Mostrar histogramas de los bundles	135
10.5.3.12	Salvar el estado de la aplicación	136
10.5.3.13	Restaurar el estado de la aplicación	137
10.5.3.14	Contar estrellas en rangos de edad y metalicidad	138
III	Conclusiones y líneas futuras	141
11	Conclusiones y líneas futuras	143
11.1	Conclusiones	143
11.2	Líneas futuras	143
11.2.1	Interfaz de línea de comandos	144
11.2.1.1	Implementación recursiva con paralelismo sobre CPU	144
11.2.1.2	Implementación: Paralelismo a nivel de CPU y de GPU	145
11.2.2	Interfaz gráfica de usuario	145
	Bibliografía	147
	Acrónimos	149
	Glosario	151
	Índice alfabético	155

Prefacio

El proyecto ha sido desarrollado para el Grupo de Investigación ‘*Evolución de galaxias en el Grupo Local*’, del Instituto Astrofísico de Canarias (IAC). Los profesores directores en la Universidad de La Laguna (ULL) del proyecto son: *Inmaculada Rodríguez Martín* y *Juan José Salazar González*. Por parte del Instituto Astrofísico de Canarias (IAC) las personas que intervienen en este proyecto son: *Carme Gallart Gallart*, *Edouard Bernard*, *Matteo Monelli*, *Lara Monteagudo Narvion* y *Antonio Dorta*. Los alumnos que llevan a cabo el desarrollo son: *Anthony Vittorio Russo Cabrera* y *Manuel Luis Aznar*.

El proyecto tiene dos objetivos principales.

El primer objetivo consiste en la implementación de un programa en C/C++ para la resolución de un problema propuesto por los investigadores del Instituto Astrofísico de Canarias (IAC) mencionados anteriormente. El programa toma como entrada dos archivos de datos, ambos de gran tamaño, y genera como salida otros dos conjuntos de archivos del mismo tamaño respectivamente. Se han realizado dos implementaciones que incluyen funcionalidades diferentes, dichas implementaciones y funcionalidades serán explicadas posteriormente en este documento. En la primera reunión los responsables del Instituto Astrofísico de Canarias (IAC) nos comentaron que tenían acceso a un programa que resolvía el problema. Sin embargo, el tiempo que tarda este programa en realizar el proceso es muy alto, lo que limita el tamaño de los ficheros que pueden usar y pensaron en la posibilidad de introducir alguna mejora, y es con este propósito con el que han contactado con la profesora *Inmaculada Rodríguez Martín*.

Los ocho primeros capítulos de este documento están dedicados al primer objetivo.

El capítulo 1 describe el problema que se pretende resolver de forma general. En el capítulo 2 se explica el problema en profundidad y se plantea una estrategia básica para su resolución.

El capítulo 3 establece el entorno que se usará para las pruebas de rendimiento y la forma utilizada para validar los resultados.

El capítulo 4 muestra los resultados de la prueba de rendimiento usando la estrategia básica.

En el capítulo 5 se explica una estrategia alternativa a la inicialmente propuesta. En los capítulos 7 y 8 se presentan dos implementaciones que usan esta estrategia para acelerar la resolución del problema. En el capítulo 6 se presentan dos opciones de paralelismo que permiten acelerar aun más si cabe la resolución del problema. En el capítulo 9 se comparan ambas implementaciones entre si.

El segundo objetivo consiste en la implementación de una interfaz gráfica de usuario (GUI) que favorezca una lectura y análisis fácil de unos ficheros de datos similares a los que se manipulan en el programa C/C++ desarrollado en el primer objetivo. En la primera reunión nos indicaron que disponían de unos scripts Python que debían ser incluidos dentro de la interfaz. Por tanto, se decidió que la interfaz sería desarrollada en Python usando PyQt para manejar los elementos gráficos.

El capítulo 10 de este documento se ha dedicado al segundo objetivo. Se comienza por una breve introducción teórica de los elementos manejados por la aplicación. Continuamos con la especificación de las herramientas utilizadas en la fase de desarrollo, una explicación muy detallada de la aplicación destinada a desarrolladores y un manual de uso destinado a los usuarios finales.

El documento concluye en el capítulo 11 mediante una serie de conclusiones personales y algunas funcionalidades nuevas que pueden ser añadidas en un futuro.

Parte I

Mejora de rendimiento

Capítulo 1

Introducción

1.1 Descripción general del problema y consideraciones

En la primera parte de este proyecto se pretende resolver el siguiente problema:

Consideremos dados dos conjuntos de datos: un **conjunto A** y un **conjunto B**.

Para cada elemento del conjunto A, se debe encontrar aquellos elementos de B que más se parecen al elemento seleccionado en A, una vez examinado todo el conjunto B y obtenidos todos los elementos semejantes, nos quedamos con uno de esos elementos semejantes de forma aleatoria. Ese elemento semejante obtenido de forma aleatoria, después de algunas transformaciones, constituye la salida del problema para el elemento seleccionado en A.

Por cada elemento del conjunto A, se genera un elemento de salida y es por ello que la salida de nuestro problema no es más que otro conjunto que llamamos, **conjunto C** y cuyo tamaño y formato es idéntico al del conjunto A. También se genera como salida un **conjunto D** de igual tamaño y formato que el conjunto B que describiremos en detalle más adelante.

Es importante hacer la siguiente consideración y es que el tamaño de los conjuntos A y B es bastante significativo. El cardinal del conjunto B suele ser menor que el cardinal del conjunto A, pero, aún así estamos tratando con conjuntos cuyo cardinal alcanza los **10 millones como mínimo**. Durante el desarrollo del proyecto se ha trabajado con los siguientes cardinales, en el **conjunto A de 16 millones** y en el **conjunto B de 12 millones aproximadamente**.

Como se ha indicado anteriormente el cardinal de **ambos conjuntos** es grande y el problema que nos ocupa es un **problema de exploración (búsqueda)** dentro de un conjunto (conjunto B), **dado un elemento de otro conjunto** (conjunto A).

Por tanto, la clave para resolver este problema en el menor tiempo posible, consiste en disponer de alguna **organización especial (estructura)** para los elementos del conjunto B, que minimice el tiempo de búsqueda. Es de suma importancia destacar que **los elementos del conjunto B no se modifican** en ningún momento durante el proceso que se lleva a cabo con el conjunto A y por tanto la estructura tampoco.

1.2 Descripción de los elementos del problema

En el apartado anterior se explico el problema de forma general, sin profundizar en el significado que tiene cada conjunto y los elementos que pertenecen a ellos. A continuación procederemos con una descripción algo más exhaustiva de los conjuntos y sus elementos.

Nos enfrentamos a la resolución de un problema en el campo de la astronomía. Uno de los elementos que se estudian en este campo son las estrellas. El problema abordado intenta corregir los errores observacionales en la fotometría introducidos por los errores del telescopio, condiciones meteorológicas, etcétera. Cada elemento perteneciente a los conjuntos A, B, C y D es una estrella que tiene una serie de datos asociados.

Cada uno de los conjuntos mencionados anteriormente son ficheros de datos, de aquí en adelante nos referiremos a ellos de la siguiente forma. El **conjunto A** es el **fichero madre**. El **conjunto B** es el

fichero crowding. Ambos ficheros constituyen la entrada de nuestro problema. El **conjunto C** es el **fichero dispersado** y por último el **conjunto D** es una copia del **fichero crowding de entrada**, pero, a diferencia de éste, está **ordenado** (de una determinada forma que explicaremos más adelante) y las estrellas tienen asociado un **dato adicional**, este fichero será el **fichero crowding de salida**. Estos dos últimos ficheros constituyen la salida de nuestro problema.

Cada línea del fichero madre representa una **estrella ‘sintética’, creada con un determinado algoritmo**¹, que reproduce un **diagrama color-magnitud**, con ciertas características a partir de **modelos de evolución estelar**. Por simplicidad las llamaremos **estrellas ‘madre’**. Cada línea del fichero dispersado corresponde a una **estrella que llamaremos dispersa**. Dado que el fichero crowding de salida es una copia del fichero crowding de entrada e incluye sólo algunos datos adicionales sólo hablaremos de **estrella crowding**.

Una estrella madre se compone de los siguientes datos **Magnitud1, Magnitud2, Edad, Metalicidad, X e Y**. Los dos primeros valores son números reales con tres cifras decimales. La Edad es un número natural expresado en notación científica de cuatro cifras significativas. La Metalicidad es un número real expresado en notación científica de cuatro cifras significativas. Los dos últimos valores son números reales con una cifra decimal. Los campos **Magnitud1** y **Magnitud2** de esta estrella son magnitudes artificiales; es decir, no tomadas por ningún telescopio y se corresponden con el filtro B y el filtro R, respectivamente del programa *IAC-STAR*.

Una estrella crowding del fichero crowding de entrada se compone de **Magnitud1, Magnitud2, Magnitud3, Magnitud4, X e Y**. Los cuatro primeros valores son números reales con tres cifras decimales. Los dos últimos valores son números reales con una cifra decimal. Una estrella crowding del fichero crowding de salida tiene los seis datos anteriores y un **número entero** (cuyo significado será especificado más adelante). Los campos **Magnitud1** y **Magnitud2** de esta estrella son las magnitudes artificiales que no tienen errores inyectados en la imagen tomada por el telescopio y se corresponden con el filtro B y el filtro R, respectivamente de la programa *IAC-STAR*. Los campos **Magnitud3** y **Magnitud4** de esta estrella son las magnitudes inyectadas **Magnitud1** y **Magnitud2**, respectivamente, recuperadas después de realizar todo el proceso del cálculo de la fotometría de las imágenes. Estas últimas dos magnitudes recuperadas si tienen errores.

Una estrella dispersada se compone de **Magnitud1’, Magnitud2’, Edad, Metalicidad, X, Y** y **tres números enteros**. Los dos primeros valores son números reales con tres cifras decimales. La Edad es un número natural expresado en notación científica de cuatro cifras significativas. La Metalicidad es un número real expresado en notación científica de cuatro cifras significativas. Los dos siguientes valores, X e Y, son números reales con una cifra decimal. Explicaremos en detalle cómo se obtienen cada uno de ellos posteriormente. Por último, los tres números enteros indican el resultado del proceso para la estrella madre seleccionada. De nuevo especificaremos cómo se obtienen y su significado posteriormente. El significado de esta estrella es obtener una estrella madre sin que esta esté afectada por errores observacionales en la fotometría.

Los datos que componen cualquiera de las estrellas se disponen en los ficheros de izquierda a derecha en el orden descrito anteriormente. Aunque se ha descrito la representación de los datos a partir del contenido de los ficheros proporcionados por los investigadores del Instituto Astrofísico de Canarias (IAC), se ha brindado soporte para múltiples representaciones numéricas.

Para obtener mayor detalle sobre este proceso y los datos utilizados, el lector puede consultar el documento: Aparicio y Gallart, «IAC-STAR: A Code for Synthetic Color-Magnitude Diagram Computation».

¹El algoritmo está implementado en Instituto Astrofísico de Canarias, *Desarrollo del Programa público IAC-STAR*.

Capítulo 2

Descripción del problema y parámetros de entrada

2.1 Especificación de los parámetros de entrada

Antes de pasar a explicar el problema en profundidad, hacen falta algunos parámetros de entrada adicionales para poder generar los ficheros de salida, a partir de los ficheros de entrada.

Como se desprende de la explicación realizada en el apartado 1.2, es imprescindible especificar cuatro nombres de ficheros, dos para los ficheros de entrada (fichero madre y fichero crowding de entrada) y dos para los ficheros de salida (fichero dispersado y fichero crowding de salida). De forma adicional y para validar los resultados obtenidos en la ejecución del proceso necesitamos otro nombre de fichero, un fichero de salida (fichero de selecciones). Sin embargo, para poder ejecutar el proceso, además de estos cinco parámetros (nombres de ficheros), se necesitan algunos parámetros más.

Las búsquedas y selecciones de estrellas dentro del fichero crowding de entrada se realizan en base a **cuatro características**. Cada característica tiene asociado un **valor de intervalo**. En el siguiente apartado explicaremos el problema y denotaremos estos valores de intervalo por la letra ε y un subíndice.

Otro parámetro de entrada necesario, es un valor entero, al que denotaremos por la letra N . Este parámetro indica el número mínimo de estrellas crowding que se deben seleccionar al realizar una búsqueda.

Por último, se necesita otro parámetro para poder realizar el proceso y al que denotaremos con el nombre de **tolerancia**.

2.2 Pseudocódigo

Una vez que hemos descrito todos los elementos y los parámetros de entrada que intervienen en el problema, podemos proceder a describir el problema en profundidad.

Cada estrella madre 'i' tiene los siguientes datos asociados **Magnitud1**, **Magnitud2**, **Edad**, **Metalicidad**, **X** e **Y**, que serán denotados como $M1_i$, $M2_i$, $Edad_i$, $Metal_i$, X_i e Y_i para una mayor simplicidad, respectivamente. Asimismo cada estrella crowding 'j' tiene **Magnitud1**, **Magnitud2**, **Magnitud3**, **Magnitud4**, **X** e **Y**, que denotaremos como $M1in_j$, $M2in_j$, $M1out_j$, $M2out_j$, X_j e Y_j respectivamente.

Para describir el problema en profundidad, a continuación, presentamos un pseudocódigo del mismo.

Entrada: fichero madre, fichero crowding entrada, ε_m , ε_c , ε_x , ε_y , N , tolerancia. **Salida:** fichero dispersado

```
1: for all estrellai del fichero-madre do
2:
3:   estrellas-seleccionadas = {}
4:   for all estrellaj del fichero-crowding do
5:     if  $M1_i - \varepsilon_m \leq M1in_j \leq M1_i + \varepsilon_m$  and
6:        $(M1_i - M2_i) - \varepsilon_c \leq M1in_j - M2in_j \leq (M1_i - M2_i) + \varepsilon_c$  and
7:        $X_i - \varepsilon_x \leq X_j \leq X_i + \varepsilon_x$  and
```

```

8:          $Y_i - \varepsilon_y \leq Y_j \leq Y_i + \varepsilon_y$  then
9:             estrellas-seleccionadas = estrellas-seleccionadas  $\cup$  estrellaj
10:        end if
11:    end for
12:
13:    if |estrellas-seleccionadas| < N then
14:
15:        estrellas-seleccionadas = {}
16:        for all estrellaj del fichero-crowding do
17:            if  $M1_i - 2 * \varepsilon_m \leq M1_{in_j} \leq M1_i + 2 * \varepsilon_m$  and
18:                 $(M1_i - M2_i) - 2 * \varepsilon_c \leq M1_{in_j} - M2_{in_j} \leq (M1_i - M2_i) + 2 * \varepsilon_c$  and
19:                 $X_i - 2 * \varepsilon_x \leq X_j \leq X_i + 2 * \varepsilon_x$  and
20:                 $Y_i - 2 * \varepsilon_y \leq Y_j \leq Y_i + 2 * \varepsilon_y$  then
21:                    estrellas-seleccionadas = estrellas-seleccionadas  $\cup$  estrellaj
22:                end if
23:            end for
24:
25:            if |estrellas-seleccionadas| < N then
26:                Escribir en fichero-dispersado la línea:
27:                     $M1_i, M2_i, Edad_i, Metal_i, X_i, Y_i, 2, 0$ 
28:            else
29:                estrella-seleccionada = random(estrellas-seleccionadas)
30:                if estrella-seleccionada.M1outj > tolerancia or
31:                    estrella-seleccionada.M2outj > tolerancia then
32:                    Escribir en fichero-dispersado la línea:
33:                        99.999, 99.999,  $Edad_i, Metal_i, X_i, Y_i, 2, |estrellas - seleccionadas|$ 
34:                else
35:                    Escribir en fichero-dispersado la línea:
36:                         $M1_i - (estrella-seleccionada.M1_{in_j} - estrella-seleccionada.M1_{out_j}),$ 
37:                         $M2_i - (estrella-seleccionada.M2_{in_j} - estrella-seleccionada.M2_{out_j}),$ 
38:                         $Edad_i,$ 
39:                         $Metal_i,$ 
40:                         $X_i,$ 
41:                         $Y_i,$ 
42:                        2,
43:                         $|estrellas-seleccionadas|$ 
44:                end if
45:            end if
46:        else
47:            estrella-seleccionada = random(estrellas-seleccionadas)
48:            if estrella-seleccionada.M1outj > tolerancia or
49:                estrella-seleccionada.M2outj > tolerancia then
50:                Escribir en fichero-dispersado la línea:
51:                    99.999, 99.999,  $Edad_i, Metal_i, X_i, Y_i, 1, |estrellas - seleccionadas|$ 
52:            else
53:                Escribir en fichero-dispersado la línea:
54:                     $M1_i - (estrella-seleccionada.M1_{in_j} - estrella-seleccionada.M1_{out_j}),$ 
55:                     $M2_i - (estrella-seleccionada.M2_{in_j} - estrella-seleccionada.M2_{out_j}),$ 
56:                     $Edad_i,$ 
57:                     $Metal_i,$ 
58:                     $X_i,$ 
59:                     $Y_i,$ 
60:                    1,
61:                     $|estrellas-seleccionadas|$ 
62:            end if
63:        end if
64:    end for

```

Podríamos sintetizar el pseudocódigo, como sigue: para cada estrella madre se realiza una primera

búsqueda y selección, si procede, de estrellas del fichero crowding basándonos en los **cuatro parámetros** ($\varepsilon_m, \varepsilon_c, \varepsilon_x, \varepsilon_y$) **que definen los intervalos**. Cada uno de los parámetros anteriores define el intervalo de la característica con la que esta asociado. Una vez se ha revisado todo el fichero crowding se comprueba si el cardinal del conjunto de estrellas seleccionadas alcanza el parámetro de entrada **N (número mínimo de estrellas crowding a seleccionar)**, si lo alcanza seleccionamos una estrella de forma aleatoria y teniendo en cuenta los valores asociados a la estrella que obtengamos generamos la salida. En caso de no alcanzar **N**, repetiremos el proceso de búsqueda y selección, pero, **multiplicando por dos los cuatro parámetros que definen los intervalos** relajando así las condiciones por las cuales las estrellas se seleccionan o no. De nuevo comprobaremos si se alcanza **N**, si es así procedemos de igual forma que al principio (escogemos una estrella de forma aleatoria y generamos la salida). En caso de que no se alcance el número mínimo **N** generamos la salida sin modificar los valores asociados a las estrellas madres, e indicamos tal circunstancia.

La **segunda** condición por la que se buscan y seleccionan estrellas (líneas 6 y 18 del pseudocódigo) consiste en **una resta de dos columnas de los ficheros madre y crowding de entrada que indican la magnitud de la estrella**. Esta operación de resta entre ambas columnas calcula el **color** de ambas estrellas respectivamente, es decir, la resta de dos valores de magnitud nos da lo que denominaremos el color de la estrella.

El proceso de seleccionar una estrella madre para generar su correspondiente salida (**estrella dispersada**), se denomina proceso de **dispersión**. Para cualquier estrella la salida puede tener dos resultados posibles, ser dispersada o no. Hay dos indicadores (números enteros) en el fichero dispersado que proporcionan información acerca del resultado de este proceso:

- El primero (**séptima columna del fichero dispersado**) indica en qué búsqueda se dispersó la estrella madre: 1 se dispersó en la primera búsqueda, 2 se dispersó en la segunda y 0 no se dispersó.
- El segundo (**octava columna del fichero dispersado**) muestra el cardinal del conjunto de estrellas seleccionadas en la búsqueda en la que se realizó la dispersión. Si no se pudo realizar la dispersión se muestra el número de estrellas que se seleccionaron en la segunda búsqueda.

En el pseudocódigo se muestra la generación del fichero dispersado. Sin embargo, hay un segundo fichero (fichero crowding de salida) y un tercer fichero (fichero de selecciones) que forman parte de la salida del programa, pero, la generación de los mismos no se muestra en el pseudocódigo por simplicidad.

- El fichero crowding de salida contiene los valores de las estrellas crowding y un número entero que indica el número de veces que se ha usado cada estrella crowding en el proceso de dispersión. Las estrellas crowding se imprimen en el fichero crowding de salida siguiendo un orden especial que será especificado en el capítulo 5.
- El fichero de selecciones será explicado en el apartado 3.2.

El pseudocódigo mostrado anteriormente tiene una complejidad de $O(n \cdot m)$, considerando que **n** es el cardinal del conjunto madre y **m** es el cardinal del conjunto crowding. Si los cardinales de ambos conjuntos son demasiado grandes, el tiempo que en la práctica puede tomar la resolución del problema de esta forma es **inasumible**, por tanto hay que buscar alguna forma para acelerar la resolución del mismo. Viendo el pseudocódigo es trivial observar que en el caso peor, por cada estrella madre se realizarán dos recorridos completos del fichero crowding. Por tanto, parece lógico pensar que cualquier **ahorro de tiempo computacional** va a depender de la **estructura y organización del fichero crowding**. La utilización de dicha estructura produce un ahorro significativo de tiempo de cómputo, pero, no suficiente debido al enorme volumen de datos de entrada, por tanto, para proporcionar una mayor aceleración se ha usado paralelismo a nivel de CPU y GPU como más adelante explicaremos.

Conviene comentar en este punto que aunque en el pseudocódigo anterior las búsquedas y selecciones de estrellas dentro del fichero crowding son realizadas atendiendo a cuatro condiciones, asimismo también se nos requirió que las búsquedas y selecciones se pudieran realizar sólo por las dos primeras condiciones, la condición de magnitud y la condición de color, excluyendo de esta forma las condiciones X e Y (líneas 7 y 8 y líneas 19 y 20).

Capítulo 3

Cuestiones previas a la implementación

3.1 Descripción y configuración utilizada para la ejecución de pruebas

Antes de la realización de cualquier prueba describiremos algunas cuestiones importantes, como las siguientes:

- Configuración hardware y software utilizada en las pruebas.
- Archivos de datos (**fichero madre** y **fichero crowding de entrada**).
- Parámetros de entrada (**características, valores de error, N y tolerancia**).
- Formato de presentación de las pruebas.

La configuración hardware y software del equipo utilizado para realizar cualquier prueba de rendimiento es la siguiente:

- Especificaciones Hardware:
 - Procesador: Intel Core i7-4930K, 3.4 GHz.
 - Memoria RAM: 32 GBytes DDR3, 1333 MHz.
 - Tarjeta Gráfica: Nvidia GeForce Titan Black, GPU GK110, 967 MHz. 2880 CUDA cores. 6 GBytes VRAM GDDR5 a 7000 MHz.
- Especificaciones Software:
 - Sistema Operativo Linux. Distribución Kubuntu v15.04
 - GCC v4.9.2
 - OpenMP 4.0
 - CUDA Toolkit 7.5

Para realizar las pruebas se han utilizado los siguientes pares de archivos de entrada:

- Primer par de archivos entrada:
 - Fichero Madre:
 - basti_0002_008_cuadrado_all_reducidoXY_nuevo2.databs
 - 1500000 líneas
 - Fichero Crowding de entrada:
 - crow_m1_nuevo2.databs

- 254418 líneas
- Segundo par de ficheros de entrada:
 - Fichero Madre:
 - lmc_johnson_reducidoXY.databs
 - 16000000 líneas
 - Fichero Crowding de entrada:
 - f10.comp.calabs
 - 11907631 líneas

Las **características utilizadas** cuando el número de condiciones es cuatro, son las siguientes (la forma de denotar los datos es la misma que en el apartado 2.2, párrafo 2):

- Característica 1:
 - Columna 1. Fichero madre (**M1**).
 - Columna 1. Fichero crowding de entrada (**M1in**).
- Característica 2:
 - Columna 1 - Columna 2. Fichero madre (**M1 - M2**).
 - Columna 1 - Columna 2. Fichero crowding de entrada (**M1in - M2in**).
- Característica 3:
 - Columna 5. Fichero madre (**X**).
 - Columna 5. Fichero crowding de entrada (**X**).
- Característica 4:
 - Columna 6. Fichero madre (**Y**).
 - Columna 6. Fichero crowding de entrada (**Y**).

NOTA: Cuando el número de condiciones es **dos**, **sólo** se usarán **las dos primeras** características anteriores.

Los **valores de intervalo** utilizados para cada una de las características son los siguientes:

- Valor de error para la característica 1:
 - $\varepsilon_{\mathbf{m}} = 0,1$
- Valor de error para la característica 2:
 - $\varepsilon_{\mathbf{c}} = 0,04$
- Valor de error para la característica 3:
 - $\varepsilon_{\mathbf{x}} = 300,0$
- Valor de error para la característica 4:
 - $\varepsilon_{\mathbf{y}} = 300,0$

NOTA: Cuando el número de condiciones es **dos**, **sólo** se usarán **los dos primeros** valores de error anteriores.

El número mínimo de estrellas (**N**) que se debe seleccionar en las búsquedas es 10.

El valor de **tolerancia** se ha establecido en 25.0.

Los ficheros de datos y parámetros de entrada fueron proporcionados por el IAC.

Las pruebas realizadas se mostrarán en forma de tabla de cuatro columnas. Cada tabla tendrá una fila de cabecera que especificará el **número de condiciones** con las que se ha trabajado, el **par de ficheros** que se ha usado y el **tipo de versión** que indica si se ha usado o no la estructura especial que acelera las búsquedas (**nm** ->no se ha usado, **avl** ->si se ha usado).

Cada fila de una tabla constituye una prueba donde:

- La primera columna indica el número de estrellas madres que intervienen en la prueba (n-primeras estrellas del fichero madre correspondiente). El nombre de esta columna será **Estrellas procesadas**.
- La **segunda columna** indica el tiempo consumido por la prueba usando sólo un núcleo del procesador. El nombre de esta columna será **SECUENCIAL**.
- La **tercera columna** indica el tiempo consumido por la prueba usando todos los núcleos del procesador (paralelismo a nivel de CPU). El nombre de esta columna será **OPENMP**.
- La **cuarta columna** indica el tiempo consumido por la prueba usando los núcleos de la tarjeta gráfica (paralelismo a nivel de GPU de la prueba). El nombre de esta columna será **CUDA**.

De esta forma una celda, muestra el tiempo de ejecución de la prueba: SECUENCIAL, OPENMP (paralelismo a nivel de CPU) ó CUDA (paralelismo a nivel de GPU) dependiendo de la columna que miremos. Los valores de tiempo mostrados en las celdas están en **segundos**.

Para una mejor comprensión e interpretación de las tablas, cada una ellas tendrá asociada una gráfica en la que se mostrarán los tiempos obtenidos sobre el número de estrellas madre procesadas.

Para realizar las ejecuciones de las distintas pruebas se ha utilizado el **comando timeout**. Este comando sirve para establecer el **tiempo límite** que un determinado comando está en ejecución. Se ha establecido un tiempo límite de **2000 segundos** para la ejecución de cualquier prueba. Una celda vacía en una tabla significa que la prueba ha alcanzado el tiempo límite. Una celda vacía en una tabla significa que la prueba ha alcanzado el tiempo límite.

3.2 Validación de los resultados obtenidos en las pruebas

Para llevar a cabo la validación de los resultados obtenidos en las pruebas se utiliza el fichero de selecciones, una versión reducida del fichero dispersado. Cada línea de este fichero representa una estrella madre a la que se le ha aplicado el proceso explicado en el apartado 2.2 y consta de tres datos (columnas):

- El **primero** es un identificador natural de estrella madre. Este identificador se corresponde con el número de línea que ocupa la estrella madre en el fichero madre.
- El **segundo** es la séptima columna del fichero dispersado.
- El **tercero** es la octava columna del fichero dispersado.

Una vez realizada la implementación del pseudocódigo del apartado 2.2, se usó el primer par de ficheros (ver apartado 3.1) y los parámetros de entrada (características, valores de error, N y tolerancia, ver apartado 3.1). Con esta configuración se realizaron dos ejecuciones, una ejecución en la que las búsquedas tenían dos condiciones y otra ejecución con cuatro condiciones. Los resultados obtenidos en ambas ejecuciones, los dos ficheros de selecciones, fueron enviados a los responsables del IAC para su validación.

En el momento en el que los resultados obtenidos obtuvieron el visto bueno por parte de los responsables del IAC, se comenzó con la realización de pruebas sistemáticas.

Para comprobar el correcto funcionamiento de futuras nuevas versiones e implementaciones, el fichero de selecciones obtenido se compara con el correspondiente fichero de selecciones validado. La nueva versión es **correcta si las dos últimas columnas de ambos ficheros no presentan diferencias**.

Capítulo 4

Implementación y tiempos obtenidos: Pseudocódigo

En el apartado 2.2 de este documento se expuso el pseudocódigo del problema a resolver, hemos realizado la implementación del mismo en lenguaje C/C++ y procedemos a mostrar los resultados de tiempo obtenidos en la ejecución, para concluir que los tiempos obtenidos no son viables, incluso usando paralelismo a nivel de CPU o GPU. Por ello, es necesario ir más allá y buscar, como ya hemos indicado, alguna estructura y organización del fichero crowding que permita acelerar las búsquedas y la resolución del problema.

A continuación mostramos una serie de tablas y gráficas donde se recogen los resultados de tiempo obtenidos al ejecutar la **implementación del pseudocódigo** del apartado 2.2. Este hecho se indica en la fila de cabecera en cada una de las tablas mediante tipo de versión ($n \cdot m$).

La primera columna (**Estrellas procesadas**) indica el número de estrellas madre que se han procesado en cada caso. Las tres columnas siguientes (**SECUENCIAL**, **OPEN-MP** y **CUDA**) recogen los tiempos obtenidos en los tres tipos de pruebas. Las pruebas de la columna SECUENCIAL se corresponden con el uso de un único núcleo del procesador. En las pruebas de la columna OPEN-MP se hace uso de todos los núcleos del procesador (paralelismo a nivel de CPU). En las pruebas de la columna CUDA se hace uso de procesadores gráficos Nvidia (paralelismo a nivel de GPU). La unidad de tiempo que se ha usado es el **segundo**.

Algunas celdas de las tablas están vacías. La razón de este hecho radica en que se ha establecido un **tiempo límite** de **2000** segundos para la ejecución de las pruebas.

La configuración de los parámetros de entrada (**características**, **valores de error**, **N** y **tolerancia**) es la misma para las cuatro tablas y puede ser consultada en el apartado 3.1.

La configuración que varía entre las cuatro tablas es la pareja de ficheros de datos utilizada y el número de condiciones por el que se realizan las búsquedas y selecciones de estrellas crowding. Las dos parejas de ficheros se pueden consultar en el apartado 3.1.

Cada tabla tiene asociada una **gráfica** que muestra los resultados de tiempo obtenidos. El **eje de abscisas** muestra el número de **estrellas madres procesadas**. El **eje de ordenadas** muestra el **tiempo de procesado** en segundos. Los resultados de los tres tipos de pruebas se muestran en las gráficas mediante líneas de distinto color. Para los resultados de la columna SECUENCIAL se utiliza una línea de color **azul**. Para los resultados de la columna OPEN-MP se utiliza una línea de color **rojo**. Para los resultados de la columna CUDA se utiliza una línea de color **verde**. Las pruebas que **excedan** el **tiempo límite** de **2000** segundos **no se mostrarán** en las gráficas.

Los resultados obtenidos en estas pruebas se usarán posteriormente con el objetivo de hacer comparaciones.

Para las pruebas recogidas en la **tabla 4.1** se han utilizado como ficheros de datos el **primer par de ficheros** cuyo **tamaño** puede ser consultado en el **apartado 3.1**. El número de condiciones que se han utilizado para realizar las búsquedas y selecciones de estrellas crowding es **2**.

En la figura 4.1 se puede observar que cuando se procesa un número de estrellas madre relativamente

• Número de Condiciones	2		
• Ficheros de datos	Primer par (Ver apartado 3.1)		
• Tipo de versión	$n \cdot m$		
Estrellas procesadas	SECUENCIAL	OPEN-MP	CUDA
10	0,0207	0,0056	0,1947
100	0,2113	0,0437	0,3153
1 000	2,0894	0,4187	0,5169
10 000	21,0414	4,1433	0,5295
100 000	210,0908	41,4489	3,4588
1 000 000	–	413,9353	36,029
1 500 000	–	621,9661	53,5233

Tabla 4.1: Rendimiento del pseudocódigo (apart. 2.2): **dos** condiciones y **primer par** de ficheros.

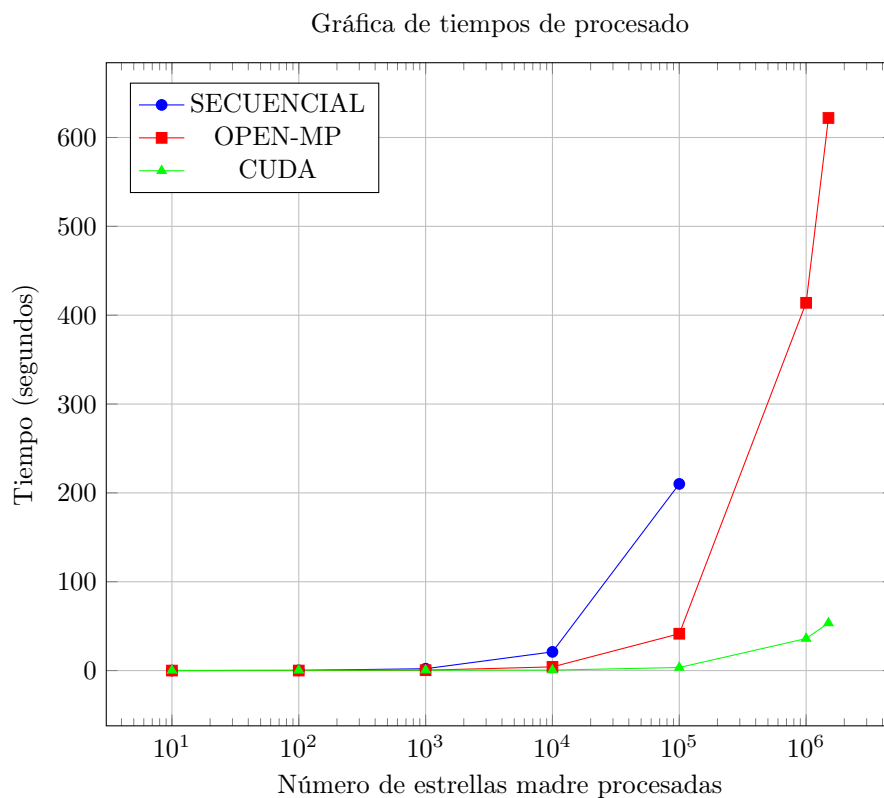


Figura 4.1: Gráfica de tiempos de procesado. Tabla 4.1.

pequeño (hasta 1000) **no existe diferencia** entre usar paralelismo (OPEN-MP y CUDA) o no usarlo (SECUENCIAL).

El punto de ruptura entre usar paralelismo o no usarlo se **manifiesta ligeramente** por primera vez al procesar 10^4 estrellas madre. Este punto de ruptura se **consolida** cuando procesamos 10^5 estrellas madre.

El punto de ruptura entre ambos tipos de paralelismo se **empieza a notar** cuando se procesan 10^5 estrellas madre. El punto de ruptura se produce **definitivamente** cuando procesamos 10^6 estrellas madre.

La figura 4.1 deja muy claro que CUDA es la ejecución ganadora en este caso.

Para las pruebas recogidas en la **tabla 4.2** se han utilizado como ficheros de datos el **primer par de ficheros** cuyo **tamaño** puede ser consultado en el **apartado 3.1**. El número de condiciones que se han utilizado para realizar las búsquedas y selecciones de estrellas crowding es **4**.

• Número de condiciones	4		
• Ficheros de datos	Primer par (Ver apartado 3.1)		
• Tipo de versión	$n \cdot m$		
Estrellas procesadas	SECUENCIAL	OPEN-MP	CUDA
10	0,0289	0,0099	0,3776
100	0,2698	0,057	0,3672
1 000	2,7983	0,5782	0,7029
10 000	27,3061	5,5983	0,7139
100 000	271,2808	54,6477	4,655
1 000 000	–	539,4219	46,8513
1 500 000	–	803,815	69,4861

Tabla 4.2: Rendimiento del pseudocódigo (apart. 2.2): **cuatro** condiciones y **primer par** de ficheros.

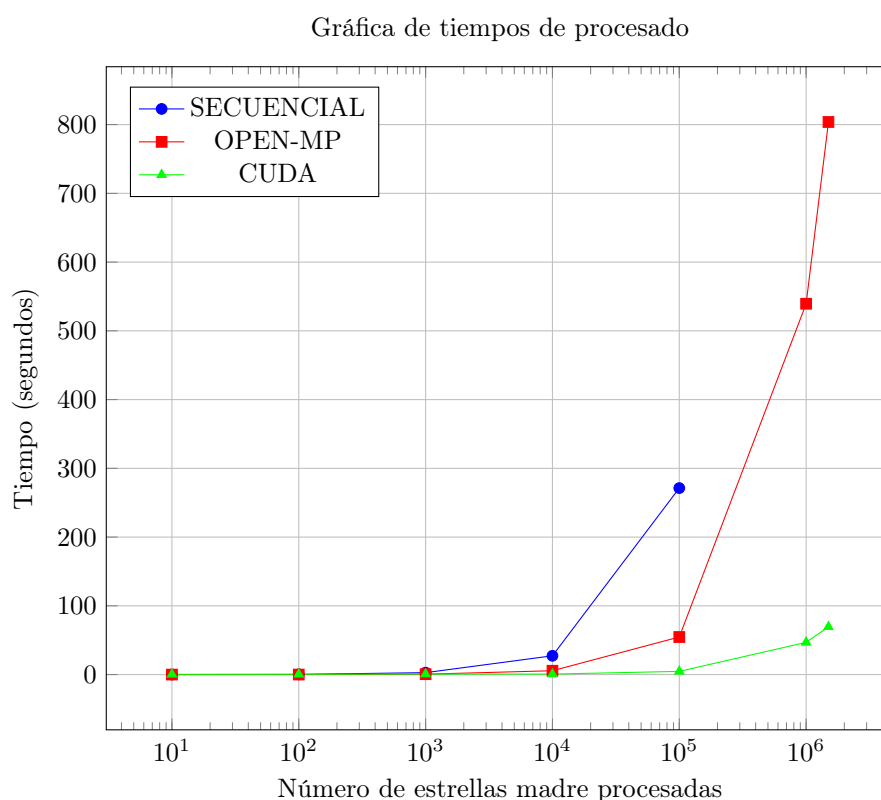


Figura 4.2: Gráfica de tiempos de procesado. Tabla 4.2.

En la figura 4.2 se puede observar la misma situación que en la figura 4.1, en cuanto a comparaciones se refiere.

La figura 4.2 apunta nuevamente que CUDA es la ejecución ganadora.

Para las pruebas recogidas en la **tabla 4.3** se han utilizado como ficheros de datos el **segundo par de ficheros** cuyo **tamaño** puede ser consultado en el **apartado 3.1**. El número de condiciones que se han utilizado para realizar las búsquedas y selecciones de estrellas crowding es **2**.

En la figura 4.3 se puede observar que cuando se procesa un número de estrellas madre relativamente pequeño (hasta 100) **no existe diferencia** entre usar paralelismo (OPEN-MP y CUDA) o no usarlo (SECUENCIAL).

El punto de ruptura entre usar paralelismo o no usarlo se **manifiesta ligeramente** por primera vez al procesar 10³ estrellas madre. Este punto de ruptura se **consolida** cuando procesamos 10⁴ estrellas madre.

Estrellas procesadas	SECUENCIAL	OPEN-MP	CUDA
10	0,9245	0,2361	7,4345
100	9,3341	1,9183	7,9466
1 000	93,8433	18,6863	14,1884
10 000	939,9352	186,2368	19,8583
100 000	–	1 859,831	129,9192
1 000 000	–	–	1 215,3894
10 000 000	–	–	–
16 000 000	–	–	–

Tabla 4.3: Rendimiento del pseudocódigo (apart. 2.2): **dos** condiciones y **segundo par** de ficheros.

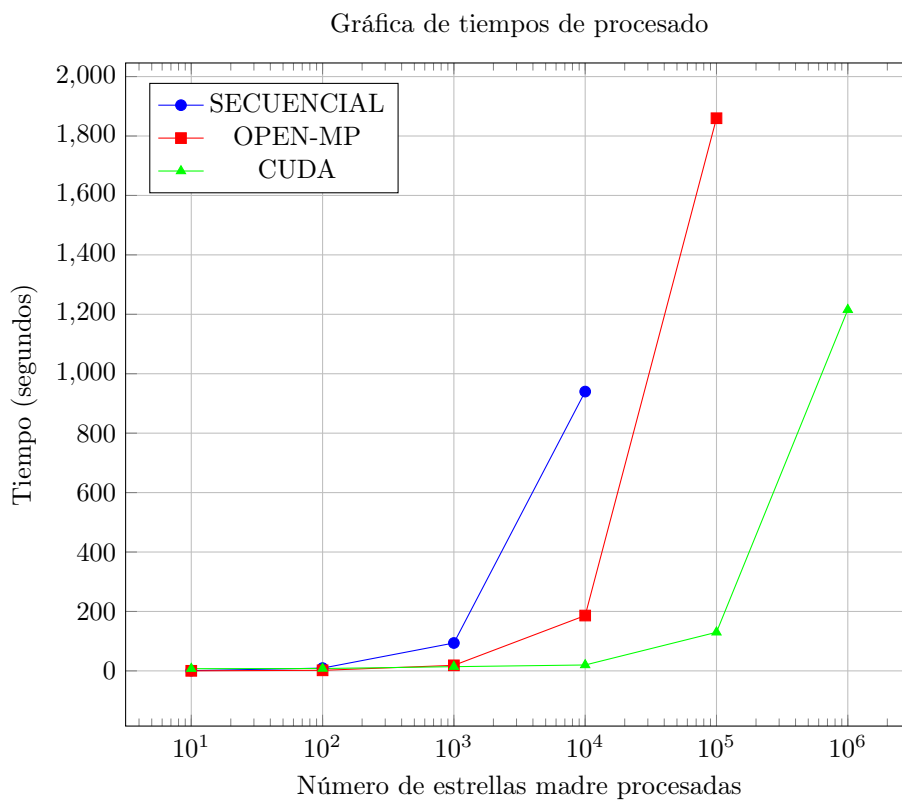


Figura 4.3: Gráfica de tiempos de procesado. Tabla 4.3.

El punto de ruptura entre ambos tipos de paralelismo se **empieza a notar** cuando se procesan 10^4 estrellas madre. El punto de ruptura se produce **definitivamente** cuando procesamos 10^5 estrellas madre.

La figura 4.3 indica que CUDA es la ganadora nuevamente.

Para las pruebas recogidas en la **tabla 4.4** se han utilizado como ficheros de datos el **segundo par de ficheros** cuyo **tamaño** puede ser consultado en el **apartado 3.1**. El número de condiciones que se han utilizado para realizar las búsquedas y selecciones de estrellas crowding es **4**.

En la figura 4.4 se puede observar la misma situación que en la figura 4.3, en cuanto a comparaciones se refiere.

La figura 4.4 apunta nuevamente que CUDA es la ganadora.

Como ya adelantamos en un primer momento y a la vista de los resultados de tiempo obtenidos al

• Número de Condiciones	4		
• Ficheros de datos	Segundo par (Ver apartado 3.1)		
• Tipo de versión	$n \cdot m$		
Estrellas procesadas	SECUENCIAL	OPEN-MP	CUDA
10	1,1139	0,2796	8,2148
100	11,6606	2,5325	15,0852
1 000	118,5543	23,4513	24,0298
10 000	1 175,1146	231,5997	25,4173
100 000	–	–	166,6097
1 000 000	–	–	1 517,0591
10 000 000	–	–	–
16 000 000	–	–	–

Tabla 4.4: Rendimiento del pseudocódigo (apart. 2.2): **cuatro** condiciones y **segundo par** de ficheros.

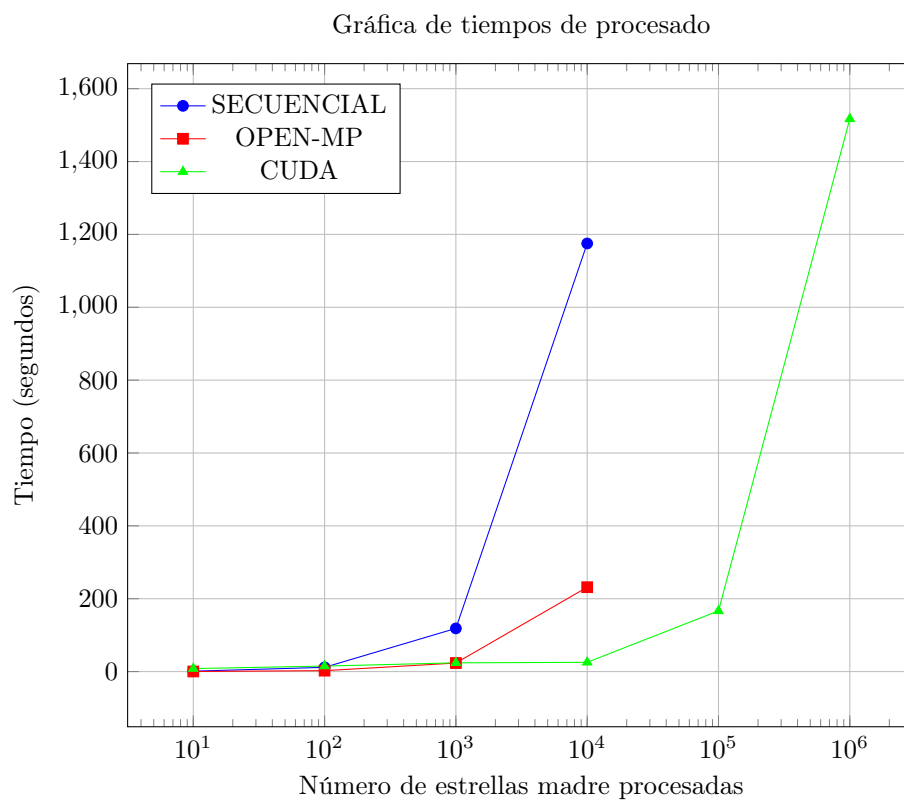


Figura 4.4: Gráfica de tiempos de procesado. Tabla 4.4.

procesar el segundo **par de ficheros**, está muy claro que la forma en la que estamos resolviendo el problema no es la adecuada, por tanto debemos buscar alguna manera de mejorar los tiempos obtenidos. Como ya habíamos adelantado, la mejora se basará en organizar y estructurar los datos contenidos en el fichero crowding de entrada de alguna forma “**inteligente**” que produzca que las búsquedas consuman menos tiempo.

Capítulo 5

Mejora del rendimiento: Estructura Índice AVL

La implementación del pseudocódigo explicado en el apartado 2.2 es la forma más simple de resolver el problema, de ahí que los resultados de tiempo obtenidos en el capítulo 4 sean tan deficientes.

El problema que nos ocupa es un problema de búsqueda dentro de un conjunto de datos, por tanto, la clave para mejorar los tiempos expuestos en el capítulo 4 radica, como ya hemos adelantado, en organizar y estructurar los datos de la mejor forma posible, consiguiendo así que el tiempo de búsqueda se reduzca considerablemente.

Recordemos que para cada estrella madre hay que seleccionar aquellas estrellas crowding que más se parezcan a la estrella madre, según cuatro características. Una estrella crowding resultará seleccionada, para una determinada estrella madre, atendiendo a una determinada característica c con error ε_c si se cumple que:

$$valor_madre_c - \varepsilon_c \leq valor_crowding_c \leq valor_madre_c + \varepsilon_c$$

En el problema inicial que plantearon los miembros del IAC, una estrella crowding resultaba seleccionada si cumplía **cuatro condiciones** como la anterior **a la vez**.

Para mejorar los tiempos de búsqueda la primera estrategia usada consistió en **ordenar el fichero crowding de entrada** por **una** de las cuatro características. Para hacer las búsquedas aplicamos la **búsqueda binaria** a la **característica ordenada**. De esta forma **reducimos el número de estrellas crowding a examinar** y el **tiempo**. Las estrellas que pasen este filtro tendrán que ser examinadas una a una para ver si cumplen con las tres condiciones restantes. Se realizó una pequeña prueba de rendimiento para comparar tiempos entre el **pseudocódigo del apartado 2.2** y esta **primera estrategia**. Los resultados obtenidos fueron:

Estrellas procesadas	Pseudoc. Apart. 2.2	Est. Búsq. Binaria
100 000	177,4042	27,0652
1 000 000	1 704,9307	247,0932
1 500 000	2 561,8171	384,2601

Tabla 5.1: Comparación de tiempos entre el pseudoc. del apart. 2.2 y la búsqueda binaria: **primer par** de ficheros.

Estrellas procesadas	Pseudoc. Apart. 2.2	Est. Búsq. Binaria
10 000	749,7087	335,0348
100 000	7 283,271	2 633,3381

Tabla 5.2: Comparación de tiempos entre el pseudoc. del apart. 2.2 y la búsqueda binaria: **segundo par** de ficheros.

Las tablas 5.1 y 5.2 muestran los tiempos consumidos por dos implementaciones. La implementación que usa la búsqueda binaria sobre una característica (Est. Búsq. Binaria) y la que no la usa (Pseudoc. Apart. 2.2). Los resultados son considerablemente diferentes. Si nos fijamos en los resultados obtenidos utilizando el primer par de ficheros de entrada podríamos caer en la tentación de quedarnos en este punto y no introducir más mejoras, pero, a la vista de los resultados obtenidos con el segundo par de ficheros de entrada parece que estos resultados aún se pueden mejorar más. Es conveniente indicar que en estas pruebas no se ha utilizado ningún tipo de paralelismo (ni a nivel de CPU, ni a nivel de GPU).

La estrategia a seguir para mejorar los resultados anteriores pasa indudablemente por poder aplicar de forma exhaustiva la búsqueda binaria en la primera característica, así como en las tres restantes (recordar que para ello tenemos que realizar ordenaciones de las mismas). Con una estructura lineal, un único vector, esto no es posible, porque, como es lógico un vector no puede ordenarse de cuatro formas (características) a la vez. La estructura básica con la que lo podemos conseguir es el **árbol**. Por tanto, utilizando el árbol como estructura básica construiremos una estructura más compleja que nos permitirá aplicar la búsqueda binaria en las cuatro características.

Hay muchos tipos de árboles, concretamente el que nosotros hemos usado es el **árbol AVL**. Hemos usado este tipo de árbol para la estructura porque, en primer lugar las operaciones de inserción y búsqueda siempre mantienen un orden de complejidad constante de $O(\log n)$ y la altura de la rama izquierda no difiere en más de una unidad a la altura de la rama derecha o viceversa.

Como ya hemos dicho se utilizara el árbol AVL como estructura básica y construiremos una estructura más compleja que nos permitirá realizar búsquedas binarias en las cuatro características. Esta estructura se muestra en la figura 5.1.

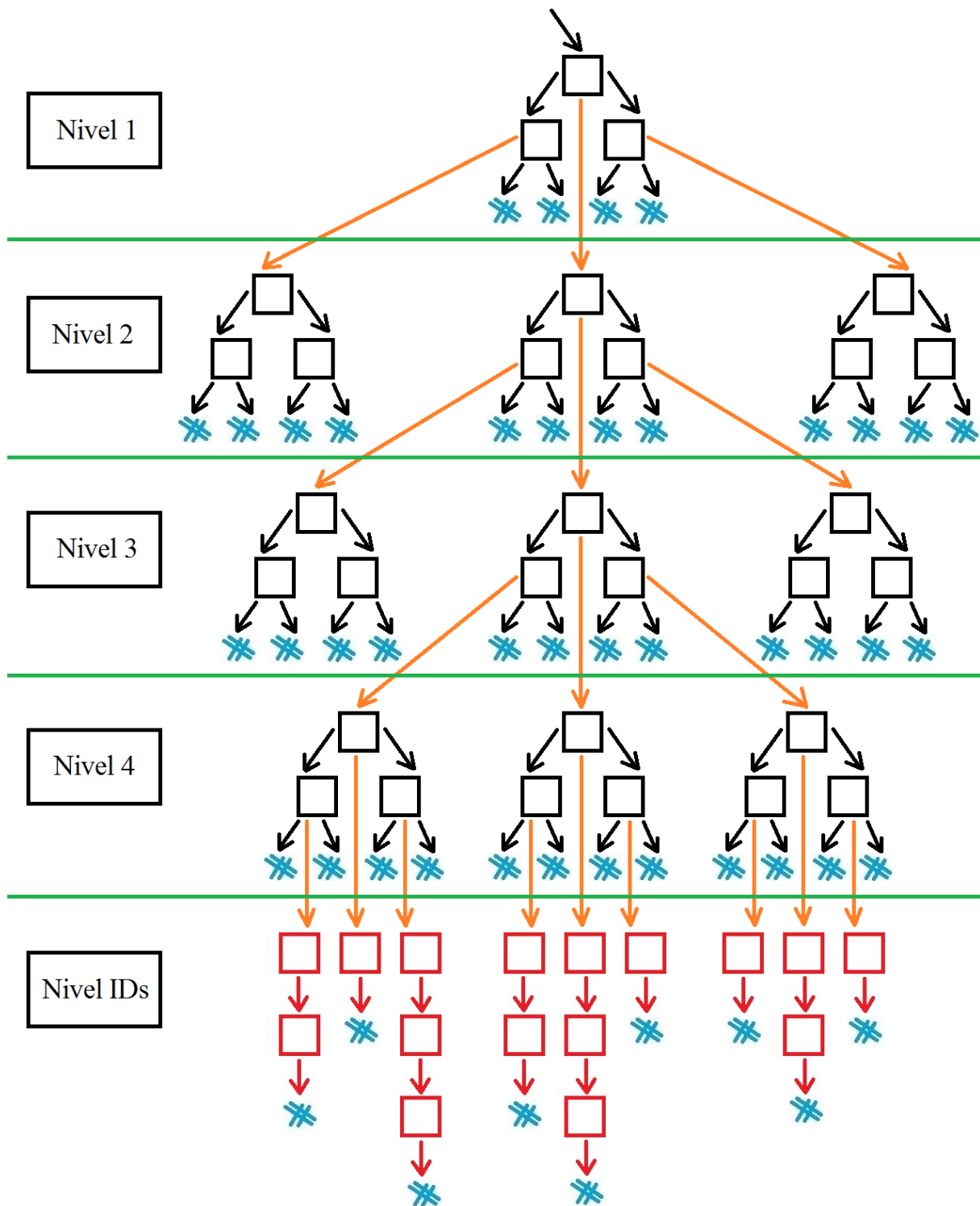


Figura 5.1: Estructura de datos utilizada para la mejora de rendimiento.

La figura 5.1 muestra la estructura que hemos utilizado para poder acelerar las búsquedas y selecciones de estrellas crowding. Empecemos entonces por explicar la estructura y los elementos que la componen:

- Cada cuadrado de la estructura (negro ó rojo) es un nodo que almacena información. Los nodos de color negro contienen información asociada a las características de las estrellas crowding. Por otra parte, los nodos de color rojo son nodos que contienen la información necesaria para identificar a las estrellas crowding de forma individual (identificadores).
- Los primeros cuatro niveles de la estructura (nivel 1 - nivel 4) almacenarán los datos de las caracte-

rísticas de las estrellas crowding. El nivel 1 almacenará los datos correspondientes a la característica 1, el nivel 2 almacenará los datos correspondientes a la característica 2, etc.

- El quinto nivel de la estructura (nivel IDs) es un nivel “especial” que almacenará los identificadores de las estrellas crowding cuyos datos relativos a características han sido almacenados en los niveles anteriores. Este nivel es necesario porque, será a través de estos identificadores, con los que podremos saber qué estrellas crowding han resultado seleccionadas después de una búsqueda y de esta forma nos será posible acceder a su información.
- Como se puede observar en la figura cada nivel está delimitado por una línea divisoria de color verde.

Las flechas que salen de los nodos representan enlaces. Hay tres tipos de enlaces:

- Los **enlaces de color negro** enlazan los nodos que contienen datos relativos a la característica del mismo nivel.
- Los **enlaces de color naranja** son enlaces entre niveles. Dado un dato de un determinado nivel el enlace indica el número de veces que se repite el dato y los datos del nivel inferior con los que está asociado el mismo. El número de asociaciones es como mínimo uno. Por simplicidad en la figura 5.1 se han omitido muchos de estos enlaces. Los **enlaces de color naranja del Nivel 4** de la figura 5.1 apuntan a aquellos identificadores de estrellas crowding cuyas características son iguales. La figura 5.1 da a entender efectivamente que hay estrellas cuyas cuatro características son iguales, no obstante, debido al grupo de cuatro características utilizado esto no será así. Cuando la estructura anterior se construye sólo teniendo en cuenta el grupo de dos características la situación mostrada en la figura 5.1 si ocurre, conviene aclarar que en este caso los niveles **tres y cuatro** mostrados en la figura 5.1 **no existen**. Los grupos de características utilizados para la construcción de la estructura se describen en el apartado 3.1.
- Los **enlaces de color rojo** enlazan los nodos que contienen los identificadores de estrellas crowding.

Para poder entender cómo se construye la estructura de la figura 5.1, expliquemos de forma simplificada el proceso que hay que llevar a cabo para insertar una estrella crowding (dados los valores de sus cuatro características) en la estructura. Se inserta el valor de la primera característica en el árbol AVL del nivel 1. La función que realiza la inserción nos devolverá un puntero a un árbol AVL del segundo nivel en el que insertar la segunda característica. Procederemos de igual forma para insertar las tres características restantes. La inserción de la cuarta (y última) característica nos devolverá un puntero a una lista de identificadores donde deberemos insertar el identificador de la estrella crowding. Cada vez que se inserta un dato en un árbol AVL de un determinado nivel se llevan a cabo, si es necesario, rotaciones para mantener la condición de equilibrio.

Una vez creada la estructura a partir del fichero crowding de entrada, se comienza con el procesado de estrellas madre. El procesado de estrellas madres **no modifica** (no se realizan inserciones, actualizaciones o borrados) la estructura.

La estructura definida en este apartado se utiliza para generar el fichero crowding de salida. Para generar este fichero recorreremos la estructura, de la misma forma en la que recorreremos un árbol AVL en inorden, e iremos escribiendo en disco los datos de cada estrella crowding junto con el indicador de uso obtenido en el proceso de dispersión.

Una vez que hemos explicado la estructura, los elementos que la componen y el proceso de creación de la misma, veamos un ejemplo con datos reales.

La tabla 5.3 muestra los valores de las estrellas crowding con los que se ha construido la estructura mostrada en la figura 5.2. Cada fila de la tabla 5.3 se corresponde con una estrella crowding:

- Un identificador que representa a la estrella crowding en cuestión y que sera almacenado en el nivel de IDs de la estructura mostrada en la figura 5.2.
- Cuatro valores (uno por característica) que serán almacenados en los niveles 1 a 4 de la estructura mostrada en la figura 5.2. Estos valores son datos reales, pero, por simplicidad se muestran como enteros.

ID Estrella Crowding	Car. 1	Car. 2	Car. 3	Car. 4
0	18	9	13	9
1	18	9	30	35
2	18	9	22	12
3	25	9	200	50
4	18	9	13	29
5	12	6	75	100
6	18	15	16	30
7	18	7	22	16
8	18	9	22	18
9	18	9	30	19
10	18	9	13	17
11	18	9	30	22
12	18	9	22	8

Tabla 5.3: Tabla de valores de estrellas crowding para construir la estructura.

La figura 5.2 muestra la estructura asociada a la tabla de valores 5.3.

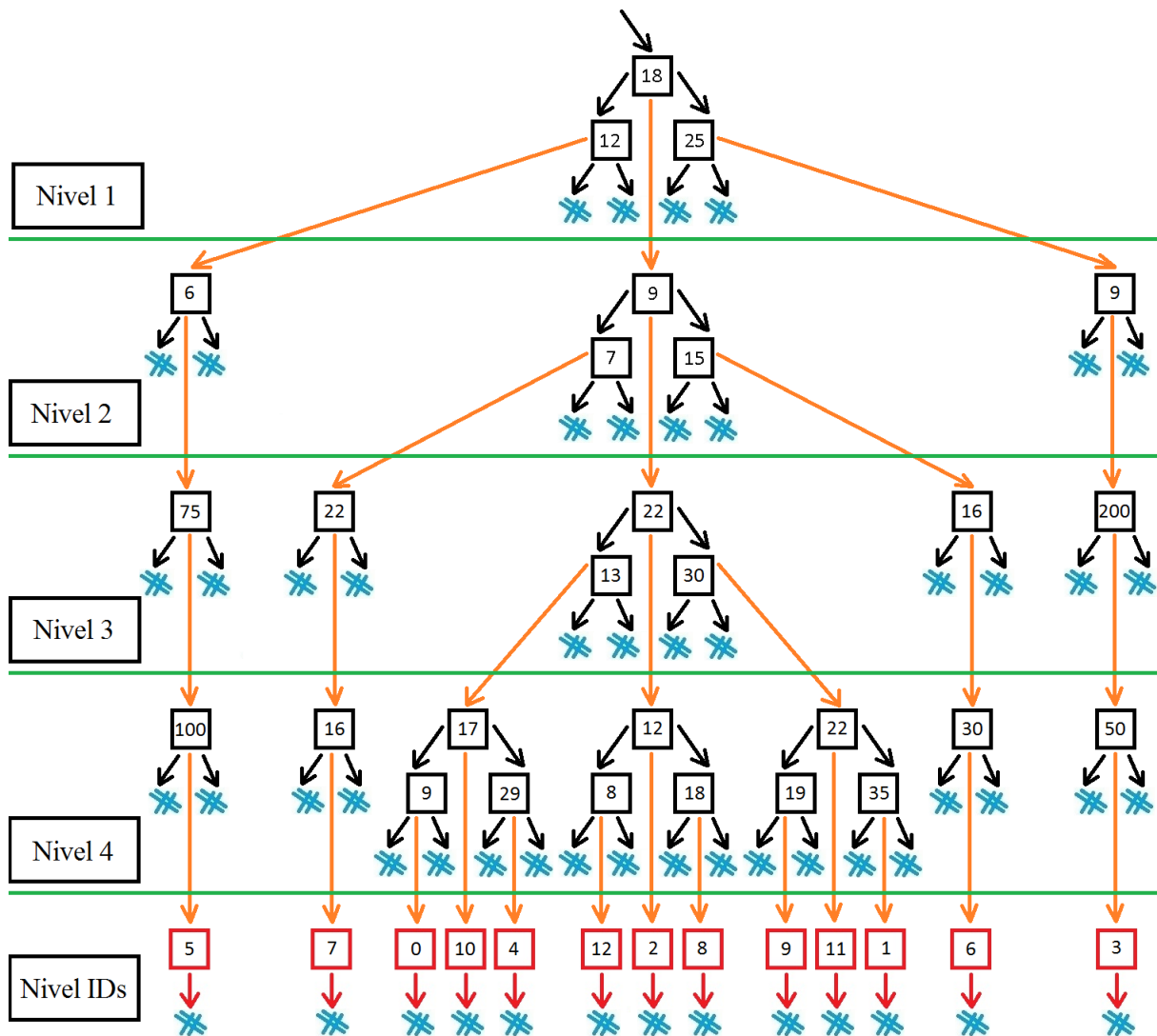


Figura 5.2: Estructura de datos correspondiente a la tabla 5.3.

La tabla 5.4 indica el tiempo que se tarda en crear la estructura para los ficheros crowding descritos en el apartado 3.1 dependiendo del número de condiciones (2 ó 4) que se utilicen.

Ficheros Crowding	Dos condiciones	Cuatro condiciones
crow_m1_nuevo2.databs	0,1133	0,1534
f10.comp.calabs	11,6601	14,8248

Tabla 5.4: Tabla de tiempos. Creación de la estructura Índice AVL.

En los capítulos 7 y 8 se explicarán las dos implementaciones, ambas se basan en esta estructura para resolver el problema de forma eficiente. Sin embargo, incluso usando la estructura que hemos explicado en este capítulo los tiempos de ejecución obtenidos para el segundo par de ficheros (ver apartado 3.1) siguen siendo demasiado altos. Por ello es que hemos usado paralelismo a nivel de CPU (OpenMP) y a nivel de GPU (CUDA). Antes de explicar las dos implementaciones, en el capítulo 6 realizaremos una breve explicación de ambos tipos de paralelismo.

Capítulo 6

Paralelismo a nivel de CPU y a nivel de GPU: OpenMP y CUDA

En este apartado se describen de forma breve los paradigmas de programación paralela utilizados para acelerar aún más el procesamiento de datos.

6.1 OpenMP: Paralelismo a nivel de CPU

OpenMP (**Open Multi-Processing**) es una API para la programación multiproceso de memoria compartida en múltiples plataformas. Permite al programador añadir concurrencia a códigos escritos en C, C++ y Fortan. Se basa en el modelo fork-join que proviene de los sistemas Unix, en el que una tarea muy pesada se divide en K hilos de menor peso (fork), al final se recopilan los resultados y se unen en uno solo (join).

OpenMP está disponible para muchas arquitecturas, incluyendo las plataformas Unix/Linux, Windows y MacOSX.

Esta API se compone de un **conjunto de directivas de compilación, rutinas de biblioteca y variables de entorno** que influyen en el comportamiento de un programa en tiempo de ejecución.

Ha sido definida de forma conjunta entre distribuidores de hardware y software. Se trata de un modelo de programación portable y escalable, capaz de proporcionar a los desarrolladores una interfaz simple y flexible para que puedan desarrollar aplicaciones paralelas para ordenadores de sobremesa y superordenadores.

La sintaxis básica de una directiva OpenMP para C/C++ es la siguiente:

```
# pragma omp nombre-directiva [cláusulas] { bloque de código }
```

Los nombres de directiva más frecuentes son **parallel**, **for**, **sections** y *combinación* de ellos.

Entre las cláusulas más frecuentes están **private**, **firstprivate**, **default**, **shared**, **copyin** y **reduction**. Todas estas cláusulas tienen que ver con la forma en la que se acceden a las variables utilizadas dentro del bloque. Por otra parte la cláusula **schedule** indica cómo se dividen las iteraciones del for entre los distintos hilos.

Cuando se incluye una directiva OpenMP esto conlleva incluir una sincronización obligatoria en todo el bloque. El bloque de código afectado se marca como paralelo, dentro del bloque trabajaran al mismo tiempo un número de hilos con las características indicadas en la directiva. Al final del bloque habrá una barrera que provocará la sincronización de los todos los hilos (salvo que se indique lo contrario mediante la directiva **nowait**). Este tipo de ejecución se denomina **fork-join**.

Para más detalles sobre directivas y cláusulas se puede consultar el link oficial de OpenMP (<http://openmp.org/>).

6.2 CUDA: Paralelismo a nivel de GPU Nvidia

CUDA (**C**ompute **U**nified **D**evice **A**rchitecture) es una plataforma de computación paralela creada por Nvidia. Permite que los desarrolladores de software utilicen las unidades de procesamiento gráfico (GPU) compatibles con CUDA que se encuentran en las tarjetas gráficas para procesamiento de propósito general. La plataforma CUDA es una capa software que permite un acceso directo al conjunto de instrucciones de la GPU y a los elementos de computación paralela para la ejecución de **kernels**.

CUDA es una plataforma que permite a los desarrolladores de software acceder a los dispositivos compatibles con CUDA mediante el uso de lenguajes de programación de alto nivel como C, C++ y Fortran. CUDA proporciona un compilador (**NVCC**) y una serie de herramientas desarrolladas por Nvidia que permiten a los programadores usar variaciones de los lenguajes anteriores para así poder codificar algoritmos en GPUs Nvidia. El objetivo consiste en facilitar el acceso de los especialistas en programación paralela a los recursos con los que cuentan las GPUs.

También es posible utilizar CUDA en Python haciendo uso de PyCUDA, así como en Java usando bindings. CUDA también soporta frameworks de desarrollo como OpenACC y OpenCL.

La versión inicial de CUDA fue publicada el 23 de Junio de 2007. La última versión estable a día de hoy es la versión 7.5 que fue publicada el 8 de Septiembre de 2015.

La compatibilidad con CUDA se puede encontrar en todas las GPUs Nvidia a partir de la serie G8X en adelante. Estas GPUs se pueden encontrar en las tarjetas Nvidia GeForce, Tesla y Quadro entre otras.

Según Nvidia, todos aquellos programas que hayan sido desarrollados para la serie GeForce 8 también funcionarán sin **necesidad de modificación alguna** en todas las tarjetas Nvidia que salgan en un futuro debido a la compatibilidad existente entre tarjetas.

CUDA explota las ventajas de los procesadores gráficos (GPUs) frente a los procesadores de propósito general (CPU). Las GPUs se componen de una serie de núcleos que permiten un paralelismo masivo siendo capaces de lanzar un altísimo número de hilos simultáneos. Aquellas aplicaciones que hayan sido diseñadas para utilizar un gran número de hilos de CPU que realizan tareas independientes también podrán usar las GPUs. Las GPUs pueden ser usadas en un gran número de campos tales como la **medicina, biología, química y un largo etcétera** donde serán capaces de ofrecer un gran rendimiento.

El modelo de programación CUDA permite a las aplicaciones escalar de forma transparente el uso de paralelismo. Este modelo contiene tres puntos claves:

- La jerarquía de grupos de hilos.
- Las memorias compartidas.
- Las barreras de sincronización.

La estructura utilizada en este modelo está definida por un grid, en el que hay una serie de bloques de hilos. Los bloques contienen como máximo 1024 hilos distintos.

Cada hilo tiene un identificador único al que se puede acceder con la variable **threadIdx**. La variable **threadIdx** tiene tres componentes (x, y, z) que coinciden con las dimensiones de bloques de hilos. Los bloques se identifican a través de la variable **blockIdx** que tiene dos componentes (x, y).

Los hilos trabajan de forma simultánea dentro de una función especial llamada **kernel**.

Un ejemplo de definición de un *kernel* en **CUDA C** es el siguiente:

```
__global__ void miKernel(args) { }
```

Para invocar el kernel anterior haríamos algo como:

```
miKernel<<gridDim, blockDim>>(args);
```

donde:

- **gridDim** es el número de instancias del kernel.
- **blockDim** es el número de threads de cada instancia.

- `args` es un número limitado de parámetros. Normalmente punteros a vectores en la memoria de la GPU y constantes copiadas por valor.

Todos los hilos trabajan de forma colaborativa unos con otros compartiendo datos dentro de un kernel, por tanto, es necesario disponer de alguna función que permita la sincronización de todos los hilos. Para ello se usa la función `__syncthreads()` que actúa como una barrera que provoca que todos los hilos esperen a que el resto terminen de ejecutar sus tareas y lleguen al mismo punto. Cuando todos los hilos han llegado a ese punto se levanta la barrera y se sigue con la ejecución de forma normal.

Los hilos pueden acceder a tres memorias diferentes:

- Cada hilo puede acceder a una zona de memoria privada.
- Cada bloque de hilos tiene una zona de memoria compartida para los hilos que forman el bloque.
- Todos los hilos pueden acceder a una memoria global.

Hay otros dos espacios de memoria de solo lectura y accesibles por todos los hilos, la **memoria de constantes** y la **de texturas**.

La implementación que resuelve el problema mediante paralelismo a nivel de GPU utilizará CUDA C. Para una mayor documentación de **CUDA C** se puede consultar el manual de programación en la siguiente URL <http://docs.nvidia.com/cuda/cuda-c-programming-guide/>.

Capítulo 7

Implementación recursiva con paralelismo sobre CPU

En este capítulo se abordan los aspectos relacionados con la implementación recursiva que utiliza OpenMP. Los aspectos más relevantes tratados son los siguientes: el objetivo de esta implementación, las herramientas utilizadas para el desarrollo, descripción detallada sobre esta implementación y finalmente la forma de utilizarla correctamente.

7.1 Introducción

El objetivo principal de esta implementación, además de resolver el problema planteado en la sección 2.2 utilizando el paralelismo de CPU, es proporcionar un marco teórico del problema que permita al usuario final ser capaz de especificar el número de índices de búsqueda que desee en el fichero de parámetros de ejecución.

Esta implementación está orientada a ser la aplicación que el equipo del IAC utilice en su entorno de producción cuando el número de condiciones de búsqueda sea diferente a dos o cuatro. Este hecho no es muy habitual, por lo que se ha decidido que esta implementación sea lo más robusta, descriptiva y versátil posible.

Entre las características destacables en esta implementación están las siguientes:

- Análisis léxico y sintáctico de los ficheros que contienen los registros de estrellas reales (madre) y artificiales (crowding).
- Generación de un fichero de registro que notifica la localización de los errores detectados en el proceso de análisis léxico y sintáctico.
- Utilizado un formato estándar y extendido para la sintaxis del fichero de parámetros. Se trata de la sintaxis JavaScript Object Notation (JSON).
- Evaluador de expresiones que permite determinar los índices de búsqueda, mediante las operaciones matemáticas entre las columnas de los ficheros que contienen los registros de estrellas reales (madre) y artificiales (crowding), sin la necesidad de que entre ellas estén relacionadas.
- Detección temprana de errores que comprueba que los datos introducidos por el usuario en el fichero de parámetros son lo suficientemente coherentes para no lanzar una ejecución estéril.
- Detección del número total de líneas en los ficheros de parámetros y contrastar dicho valor con los que el usuario introdujo, preguntando a este la posibilidad de procesar las líneas detectadas en el caso en el que estos valores difieran.
- Maximizar el consumo responsable de memoria Random Access Memory (RAM) por parte de la aplicación para ejecutar el máximo número de estrellas reales (madre) en paralelo, sin agotar los recursos del sistema, evitando disponer de la memoria de intercambio Swap, que repercutiría negativamente en el rendimiento de la ejecución.

- Supervisión de la correcta liberación de los recursos solicitados al sistema, haciendo uso de la herramienta Valgrind.
- Proporcionado un conjunto de pruebas unitarias para verificar el funcionamiento de los diferentes módulos con la intencionalidad de imprimir en el software entregado atributos de calidad.

Se han utilizado herramientas de terceros con licencias que así lo permiten, que facilitaron cumplir estos objetivos adicionales.

7.2 Herramientas de desarrollo

Un aspecto importante en el desarrollo de aplicaciones es conocer qué requisitos tienen, las herramientas utilizadas en el desarrollo y su configuración.

Esta información facilitará la labor de quienes quieran optimizar, resolver problemas, añadir nuevas funcionalidades y depurar el código fuente. Es por eso que se detalla a continuación.

7.2.1 Requisitos

En esta implementación se utilizó C++ como lenguaje de programación base, porque es lenguaje de propósito general muy extendido, robusto y fiable. Otro motivo por el que se eligió este lenguaje es que existen multitud de herramientas de terceros desarrolladas que facilitan su integración con este lenguaje.

Además de ser un lenguaje compilado, pone a disposición de los desarrolladores una multitud de funcionalidades de interés para esta implementación.

Se utilizó el estándar C++2011, comúnmente llamado, *C++11* de librerías, publicado en ISO, *ISO/IEC 14882:2011 Information technology — Programming languages — C++*.

7.2.1.1 Compilador

El principal requisito que debe cumplir la persona o equipo que desee utilizar el código fuente es instalar un compilador compatible con *C++11* y OpenMP.

7.2.1.2 Librerías y herramientas de terceros

Otro elemento indispensable en el desarrollo fue el uso de librerías y herramientas de terceros que permitieron añadir funcionalidades a la aplicación. Las librerías y herramientas que se usaron se listan a continuación, especificando en qué funcionalidad intervinieron:

- **JsonCpp**: esta librería principalmente permite utilizar la sintaxis JSON para especificar el fichero de parámetros. Además, permite obtener los datos de dicho fichero de forma rápida y cómoda.
- **ANother Tool for Language Recognition (ANTLR)**: esta herramienta permitió generar con una sintaxis fácil de entender un analizador léxico y sintáctico que es el encargado de detectar y registrar errores en el registro de estrellas.
Esta herramienta tiene un soporte limitado y experimental para generar los analizadores en C++. Se ha detectado un *bug* en el analizador generado y se ha corregido. Estos detalles se mencionarán en la sección 7.2.3.1.
- **The C++ Mathematical Expression Toolkit Library (ExprTk)**: esta librería permitió obtener de forma dinámica, a través de expresiones matemáticas contenidas en el fichero de parámetros, los valores utilizados tanto en el proceso de creación de la estructura como en el proceso de búsqueda.
- **Google C++ Testing Framework**: este Framework permitió desarrollar los test unitarios en C++.

7.2.1.3 Entorno de desarrollo

A pesar de que en esta sección se describen los requisitos, es necesario mencionar que el uso de un Integrated Development Environment (IDE) es opcional, pero altamente recomendado porque agiliza en gran medida el proceso de desarrollo de cualquier aplicación.

Entre las virtudes de utilizar esta herramienta, podemos encontrar las siguientes: automatización de tareas rutinarias como compilación, auto-completado de texto, información sobre la documentación de rutinas de la Application Programming Interface (API) utilizada, brida herramientas para la depuración, marcado de los errores detectados tanto de sintaxis como de compilación, etc.

Para el desarrollo de esta implementación se utilizó un IDE, llamado Eclipse orientado al desarrollo de aplicaciones en *C/C++*; también conocido como *C/C++ Development Toolkit (CDT)*.

Otra herramienta utilizada en el desarrollo de esta implementación es un control de versiones. Aunque su uso no supone un requisito, ofrece múltiples beneficios bien conocidos por los desarrolladores de software.

El control de versiones utilizado en el desarrollo de esta implementación es Git, junto con un repositorio remoto privado alojado en la siguiente url: <https://bitbucket.org/ULLfypIAC/crowding-effects-cli-recursive>. Si se desea obtener privilegios de lectura sobre este repositorio, contactar con cualquiera de los autores de esta memoria.

7.2.2 Instalación de herramientas

El siguiente paso es especificar de forma genérica la forma de obtener las herramientas y librerías especificadas en la sección 7.2.1.

■ Compilador

En el desarrollo de esta implementación se ha utilizado el compilador por excelencia instalado en sistemas GNU/Linux, el GNU Compiler Collection (GCC).

La versión utilizada de este compilador fue *5.2.1 20151010*, bajo el paquete *gcc-5 5.2.1-2ubuntu2* para la distribución Ubuntu 15.10 (wily werewolf).

El proceso de instalación del GCC a partir del código fuente está fuera de los objetivos de este documento. El lector que desee obtener más información al respecto puede consultar los siguientes enlaces:

- <https://gcc.gnu.org/install/>.
- <http://manualinux.heliohost.org/gcc.html>.

Existen otros compiladores alternativos al GCC. Los más conocidos, a parte del ya citado, son los siguientes: Clang y Intel C++ Compiler. Se podrá elegir cualquiera de los nombrados, asegurando que dicha versión ofrezca soporte para *C++11* y OpenMP.

■ Eclipse - CDT

Para determinar el entorno de desarrollo utilizado, es necesario especificar dos elementos: Eclipse Platform y la característica que permite el desarrollo de aplicaciones en *C/C++*, es decir, CDT.

El Eclipse Platform utilizado tiene las siguientes características:

- *Version*: 4.5.2.v20160212-1500.
- *Build id*: M20160212-1500.

La versión utilizada del plugin CDT es 8.8.1.201602051005.

Este entorno de desarrollo integrado se puede obtener en la web oficial con el siguiente enlace: <http://www.eclipse.org/cdt>.

■ JsonCpp

La versión usada es la *1.4.4*, liberada en febrero de 2015. Se puede obtener en el siguiente enlace: <https://github.com/open-source-parsers/jsoncpp/releases/tag/1.4.4>

- **ANTLR**

La versión usada es la 3.5.2, liberada en marzo de 2014. Se puede obtener en el siguiente enlace: <http://www.antlr3.org/download/antlr-3.5.2-complete.jar>

- **ExprTk**

Esta librería no tiene publicada sus números de versiones. Los enlaces disponibles para obtener ExprTk son los siguientes: <http://www.partow.net/programming/exprtk/> y <https://github.com/ArashPartow/exprtk>.

- **Google C++ Testing Framework**

La versión utilizada es la 1.7.0, liberada en septiembre de 2013. Se puede obtener en el siguiente enlace: <https://github.com/google/googletest/releases/tag/release-1.7.0>

7.2.3 Configuración de herramientas

Una vez garantizada la obtención de las herramientas necesarias, es el turno de configurarlas.

El lector que no desee modificar el código de esta implementación o depurarlo o simplemente generar un archivo ejecutable para su sistema nativo, puede omitir la lectura de esta sección.

7.2.3.1 Generar librerías estáticas

Un paso crucial es la generación de las librerías estáticas específicas del sistema operativo. Para generarlas, se utilizó la documentación propia de cada una de las herramientas de terceros utilizadas, excepto el analizador léxico y sintáctico.

Normalmente este proceso consiste en la compilación del código fuente sin invocar al *linker* del compilador¹, acto seguido utilizar la herramienta *ar*² del paquete *GNU Development Tools*.

Para generar la librería estática del analizador léxico y sintáctico es necesario que el sistema operativo disponga de Java y ejecutar un simple proceso, descrito a continuación:

1. Dirigirse al directorio `/cli/cpu/includes/analyzers/` en una terminal.
2. Ejecutar el comando `make` en dicho directorio para generar el analizador en *C++* e *intentar* compilarlo. Se ha resaltado la palabra *intentar*, porque el proceso de compilación fallará, debido a un *bug* detectado, como se ha dicho en ocasiones anteriores.
3. Editar los ficheros generados, añadiendo la palabra `const` en la declaración de las variables afectadas por el fallo y guardando los cambios.
4. Continuar el proceso de generación de la librería estática con el comando `make continue`.

Existen mensajes que informan al usuario de este proceso. Además, este proceso asume la ubicación de la herramienta y de las cabeceras de ANTLR. Para más información consultar el fichero `/cli/cpu/includes/analyzers/Makefile`.

7.2.3.2 Objetivos de compilación

En el desarrollo de software es habitual definir diferentes objetivos, conocidos con su término inglés *target*; que permiten diferenciar distintos entornos de ejecución.

Asumiendo que se ha instalado el compilador y el IDE Eclipse CDT, se puede importar el proyecto software asociado a esta implementación para recrear la mayoría de la configuración utilizada en el proceso de desarrollo. La forma de importar el proyecto es la siguiente: seleccionar el directorio

`/cli/cpu/`

siguiendo las indicaciones

File ▷ *Import* ▷ *General* ▷ *Exist Project into workspace*.

A continuación, se detalla cómo configurar el entorno de desarrollo con los diferentes *targets*, asumiendo que el proyecto software asociado a esta implementación ha sido importado.

¹Opción `-c` en el compilador `gcc`.

²Consultar `man 1 ar` para más información.

Objetivo de depuración este entorno tiene el objetivo de depurar la aplicación desarrollada.

Los pasos a seguir son los siguientes:

1. Activar el *target Debug*:
Project ▷ *Properties* ▷ *C/C++ Build* ▷ *Configuration*.
2. Excluir directorios: los directorios
/cli/cpu/sources/test,
/cli/cpu/includes/analyzers/generated y
/cli/cpu/includes/analyzers/sources
 deben ser excluido siguiendo las pautas siguientes:
 - a) hacer clic derecho sobre el directorio a excluir.
 - b) *Resource* ▷ *Configurations* ▷ *Exclude from Build...*
 - c) marcar el *target* activo.
3. Habilitar *indexer*: marcar la casilla que habilita el *indexer*, la que permite definir configuración específica al proyecto y la habilitación del almacenamiento en el propio proyecto, localizados en
Project ▷ *Properties* ▷ *C/C++ General* ▷ *Indexer*.
4. Añadir símbolo *C++11*: agregar un nuevo símbolo *__cplusplus* con valor *201103L* bajo
Project ▷ *Properties* ▷ *C/C++ General* ▷ *Paths and Symbols* ▷ *Symbols*.
5. Incluir cabeceras: agregar los directorios
/cli/cpu/includes/,
/cli/cpu/includes/exprtk y
/cli/cpu/includes/antlr3.5.2
 bajo
Project ▷ *Properties* ▷ *C/C++ Build* ▷ *Settings* ▷ *GCC C++ Compiler* ▷ *Includes* ▷ *Include paths*.
6. Incluir librerías: agregar las librerías y sus directorios de la siguiente forma:
 - a) incluir los directorios
/cli/cpu/libraries/linux_amd64/json y
/cli/cpu/libraries/linux_amd64/analyzers
 bajo
Project ▷ *Properties* ▷ *C/C++ Build* ▷ *Settings* ▷ *GCC C++ Linker* ▷ *Libraries* ▷ *Library search paths*.
 - b) incluir las librerías estáticas *jsoncpp* y *analyzers* bajo
Project ▷ *Properties* ▷ *C/C++ Build* ▷ *Settings* ▷ *GCC C++ Linker* ▷ *Libraries* ▷ *Libraries*.
7. Activar nivel de depuración: modificar el campo *Debug Level* al valor *Maximum (-g3)* bajo *Project* ▷ *Properties* ▷ *C/C++ Build* ▷ *Settings* ▷ *GCC C++ Compiler* ▷ *Debugging*.
8. Misceláneas del compilador: añadir las siguientes opciones bajo
Project ▷ *Properties* ▷ *C/C++ Build* ▷ *Settings* ▷ *GCC C++ Compiler* ▷ *Miscellaneous* ▷ *Other flags*.
 - a) *-c*: permitir la compilación de objetos sin invocar al *linker*. Necesario para las variables *export*.
 - b) *-std=c++11*: indicar al compilador el uso de las librerías estándar *C++11 ISO, ISO/IEC 14882:2011 Information technology — Programming languages — C++*.
 - c) *-fopenmp*: indicar al compilador el uso de directivas *OpenMP*.
9. Misceláneas del *linker*: añadir las siguientes opciones bajo
Project ▷ *Properties* ▷ *C/C++ Build* ▷ *Settings* ▷ *GCC C++ Linker* ▷ *Miscellaneous* ▷ *Linker flags*.
 - a) *-std=c++11*
 - b) *-fopenmp*

Objetivo de testeo este entorno tiene el objetivo la creación y ejecución de los test unitarios, que permitieron conocer si cada funcionalidad desarrollada se comportaba de la forma esperada ante una entrada conocida.

Para facilitar este proceso, se puede realizar una copia del *target Debug* ya que tienen cosas en común. Los pasos a seguir son los siguientes:

1. Crear el *target Test*: dirigirse a *Project* ▷ *Properties* ▷ *C/C++ Build* ▷ *Configuration* ▷ *Manage Configurations...* ▷ *New...* y realizar los siguientes pasos:
 - a) el campo **Name** debe tener el valor **Test**.
 - b) el campo **Description** debe tener el valor **Unit Test**.
 - c) seleccionar en **Copy setting from** la opción **Existing configuration** con el valor **Debug**.
2. Activar el *target Test*: similar paso 1 del *target Debug*.
3. Excluir directorios y ficheros: similar paso 2 del *target Debug*, con la salvedad que no se desea excluir el directorio `/cli/cpu/sources/test` y sí se desea excluir el fichero `/cli/cpu/sources/run.cpp`³.
4. Habilitar `indexer`: igual al paso 3 del *target Debug*.
5. Añadir símbolo `C++11`: igual al paso 4 del *target Debug*.
6. Incluir cabeceras: similar paso 5 del *target Debug*, sólo añadiendo el directorio `/cli/cpu/includes/`.
7. Incluir librerías: agregar las librerías y sus directorios de la siguiente forma:
 - a) incluir los directorios `/cli/cpu/libraries/linux_amd64/json` y `/cli/cpu/libraries/linux_amd64/gtest` bajo *Project* ▷ *Properties* ▷ *C/C++ Build* ▷ *Settings* ▷ *GCC C++ Linker* ▷ *Libraries* ▷ *Library search paths*.
 - b) incluir las librerías estáticas `jsoncpp` y `pthread` y `gtest` bajo *Project* ▷ *Properties* ▷ *C/C++ Build* ▷ *Settings* ▷ *GCC C++ Linker* ▷ *Libraries* ▷ *Libraries*.
8. Misceláneas del compilador: igual al paso 8 del *target Debug*.
9. Misceláneas del *linker*: igual al paso 9 del *target Debug*.

Objetivo de lanzamiento este entorno tiene el objetivo generar la aplicación usada a nivel de producción.

Su configuración es similar a la del *target Debug*, con la excepción del paso 7, que puede ser omitida.

7.3 Implementación de la aplicación

Es el momento de conocer los aspectos técnicos de esta implementación. En esta sección se abordan este tipo de detalles.

Conviene recordar, como se nombró en la sección 7.2.1, que esta implementación ha utilizado como lenguaje básico de programación el `C++` con el uso de OpenMP y el estándar de librerías de `C++11` (ISO, *ISO/IEC 14882:2011 Information technology — Programming languages — C++*).

³Esto evitará el problema de múltiples funciones `main(...)`

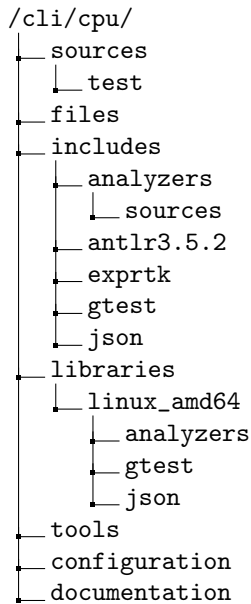


Figura 7.1: Árbol de directorios de la implementación con paralelismo sobre CPU.

7.3.1 Árbol de directorios

Este documento, como se ha mencionado en ocasiones anteriores, va acompañado de un soporte de almacenamiento, cuyo contenido es el código fuente de las implementaciones que se abordan en este documento.

Es importante conocer el árbol de directorios utilizado en esta implementación, a pesar de lo sugerente que son sus nombres.

El directorio raíz de esta implementación se encuentra bajo el directorio `/cli/cpu/`. El árbol de directorios básico de esta implementación se ilustra en la figura 7.1.

A continuación se detallan los directorios:

- **sources**: directorio donde se ubican los códigos fuente de esta implementación.
- **sources/test**: directorio que almacena los códigos fuente de las pruebas unitarias de esta implementación.
- **files**: directorio que contiene los ficheros utilizados para la ejecución del programa principal de esta implementación y también lo utiliza el programa principal de las pruebas unitarias para generar y cargar un fichero con sintáxis JSON.
- **includes**: en este directorio deben ubicarse las cabeceras de las herramientas utilizadas (ANTLR, JSON, ExprTk y Google C++ Testing Framework).
- **includes/analyzers**: contiene los ficheros fuente para generar el analizador léxico y sintáctico que analizará los posibles fallos que puedan tener los ficheros de entrada. Para generar los analizadores se ha proporcionado un fichero `Makefile`, que permite generar los códigos fuentes con ANTLR y compilarlos para crear una librería estática. Este proceso se explica con mayor detalle en la sección 7.2.3.1 y sección 7.3.6, debido a un *bug* detectado en la herramienta ANTLR.
- **libraries**: este directorio debe contener las librerías estáticas del sistema operativo con el que utilice el código fuente. Por lo que debe contener tantos directorios como sistemas operativos de familias diferentes. En este caso si utilizó el nombre `linux_amd64`.

Con la finalidad de mantener separada las librerías, se decidió ubicar tres directorios dentro de cada uno de los anteriores directorios, sin importar el nombre que tenga. Esto tres directorios (*analyzers*, *gtest* y *json*) contienen las librerías estáticas generadas: `lib*.a`.

- **tools:** en este directorio ofrecen las herramientas y librerías externas, mencionadas en la sección 7.2.1.2, con el objetivo principal de facilitar su obtención en el caso de que los enlaces no estén disponibles o no se disponga de conexión a internet para descargarlos.
- **configuration:** este directorio ofrece algunos ficheros de configuración del IDE Eclipse CDT, que faciliten la labor de configuración, así como proporcionar plugins para este entorno de desarrollo.
- **documentation:** en este directorio está alojada documentación de sobre las herramientas y librerías utilizadas.

Los ficheros y directorios de configuración IDE Eclipse CDT para este proyecto se encuentran ocultos en las siguientes rutas:

```
/cli/cpu/.project,  
/cli/cpu/.cproject y  
/cli/cpu/.settings.
```

Los ficheros y directorios de configuración Git para este proyecto se encuentran ocultos en las siguientes rutas:

```
/cli/cpu/.git y  
/cli/cpu/.gitignore.
```

Los directorios donde se encuentran los ejecutables de cada uno de los *tagets*, descritos en la sección 7.2.3.2, se encuentran en las rutas: `/cli/cpu/Release`, `/cli/cpu/Debug` y `/cli/cpu/Test`.

7.3.2 Estructura de datos

Como ya se introdujo en el capítulo 5, es necesario utilizar estructuras de datos eficientes que permitan reducir de forma significativa las operaciones de inserción y búsqueda sobre los índices definidos en el fichero de parámetros. En concreto se mencionó los árboles Adelson-Velsky Landis (AVL).

La figura 7.2 ilustra de forma parcial un árbol AVL de 4 índices anidados, también llamado niveles anidados o características anidadas.

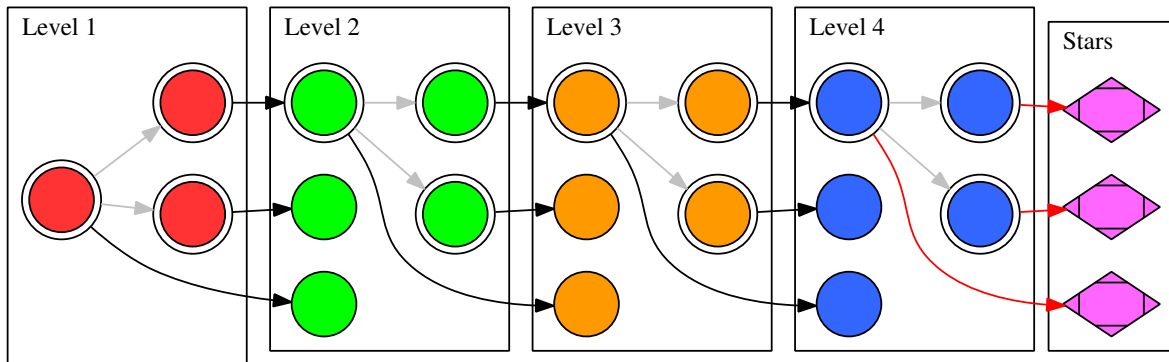


Figura 7.2: Estructura parcial de 4 niveles usando árboles AVL.

Por simplicidad, se han omitido los enlaces de los nodos coloreados de rojo, verde, naranja y azul sin doble círculo de dicha figura. En la sección 7.3.3 se detalla en con mayor precisión los datos usados en esta implementación.

La principal bondad que ofrece el uso de la estructura de árbol balanceado, es que la cota superior asintótica para todas las operaciones es de $O(\log n)$. Pero otro beneficio que ofrece esta estructura de datos es que se puede adaptarse en tamaño a cualquier magnitud de datos sin que esto suponga un esfuerzo mayor.

Ésta última propiedad (la de crecer bajo demanda sin inconvenientes) es muy adecuada para el problema abordado, ya que *a priori* no se puede predecir el número de índices que serán diferentes y por ende no se puede predecir el tamaño del árbol.

Sin embargo, a pesar de estos beneficios, la estructura de árbol balanceado presenta un desventaja en la práctica: las búsquedas son más lentas frente a las búsquedas en listas continuas en memoria RAM.

Teóricamente, ambas estructuras de datos presentan la misma cota superior asintótica para la operación de búsqueda $O(\log n)$, pero en la práctica una lista continua en memoria reduce los tiempos de búsqueda en comparación con la estructura de árbol balanceado.

Por el contrario, una estructura de lista continua en memoria no se adapta tan bien como una estructura de árbol balanceado ante una cantidad desconocida de elementos a añadir.

Por estas razones, se utilizaron ambas estructuras de datos, cada una para el contenido que mejor cumplen respecto a la otra. La estructura de árbol AVL se empleó para crear un índice anidado a medida que se leen los datos de entrada. La estructura de lista continua en memoria RAM se utilizó para realizar las búsquedas binarias, reduciendo los tiempos de búsqueda.

La creación de la lista continua en memoria RAM se realiza a partir del árbol AVL cuando finalice el proceso de lectura de los datos de entrada que conforman el índice para las búsquedas. La figura 7.3 ilustra el resultado obtenido de transformar la estructura de árbol anidado AVL en una estructura de lista anidada continua.

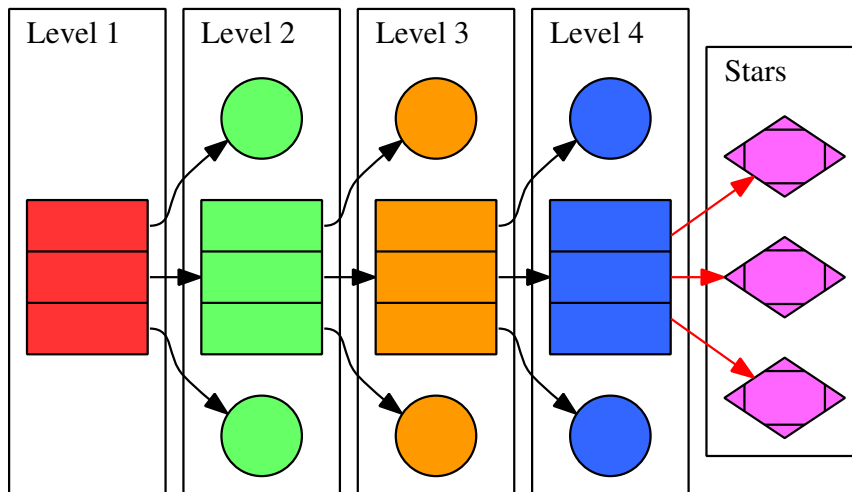


Figura 7.3: Estructura parcial de 4 niveles usando listas.

Por simplicidad, se han omitido los enlaces de los nodos coloreados de verde, naranja y azul con forma circular de dicha figura. En la sección 7.3.3 se detalla en con mayor precisión los datos usados en esta implementación.

7.3.3 Diagrama de clases

Con la intención de ofrecer una visión global de esta implementación, en esta sección se muestran los diagramas de las clase utilizadas con la notación Unified Modeling Language (UML).

Los diagramas mostrados son el fiel reflejo del código fuente adjunto, ya que se han obtenido directamente de él mediante el uso de la herramienta `cpp2dia`⁴. Además, se ha sometido a revisión para descartar posibles errores.

⁴Para más información, visitar la web oficial: <http://cpp2dia.sourceforge.net/>.

7.3.3.1 ArtificialStar

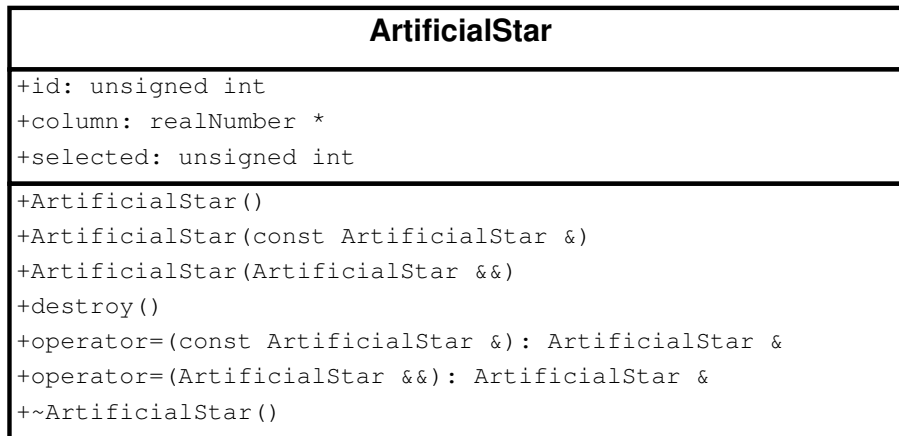


Figura 7.4: Diagrama de la clase `ArtificialStar` con notación UML.

Esta clase, ilustrada en la figura 7.4, representa una única estrella artificial, también llamada estrella crowding.

La información más relevante de esta clase es la siguiente:

- `id`: valor numérico que indica el número de fila donde se encuentra la estrella en cuestión.
- `column`: lista de valores numéricos con representación en coma flotante, que contienen cada una de las columnas leídas del fichero.
- `selected`: valor numérico que representa el número de veces que esta estrella fue seleccionada de forma aleatoria entre el conjunto de estrellas buscada en el proceso de dispersión

En realidad este dato no es una clase del lenguaje *C++*, sino que es una `struct` avanzado de *C++11*, que no sólo tiene atributos, sino que también dispone de métodos especiales como constructor, destructor, etc. Por eso a los efectos, se comporta como una clase.

7.3.3.2 RealStar

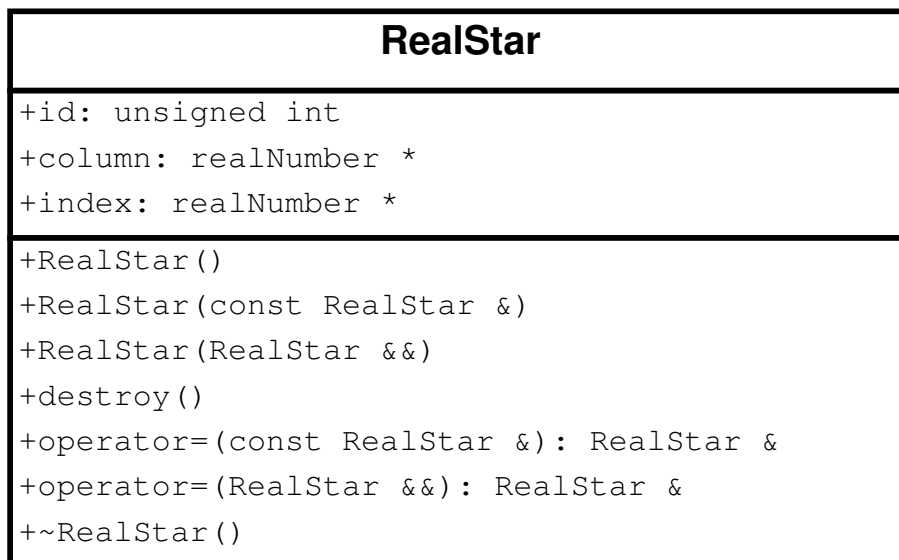


Figura 7.5: Diagrama de la clase `RealStar` con notación UML.

Esta clase, ilustrada en la figura 7.5, representa una única estrella real, también llamada estrella madre o estrella sintética.

La información más relevante de esta clase es la siguiente:

- **id**: valor numérico que indica el número de fila donde se encuentra la estrella en cuestión.
- **column**: lista de valores numéricos con representación en coma flotante, que contienen cada una de las columnas leídas del fichero.
- **index**: lista de valores numéricos con representación en coma flotante, que son utilizados en el proceso de dispersión para la búsqueda de estrellas artificiales «similares».

Este tipo de dato se trata de un **struct**, al igual que la anterior.

7.3.3.3 DispersedStar

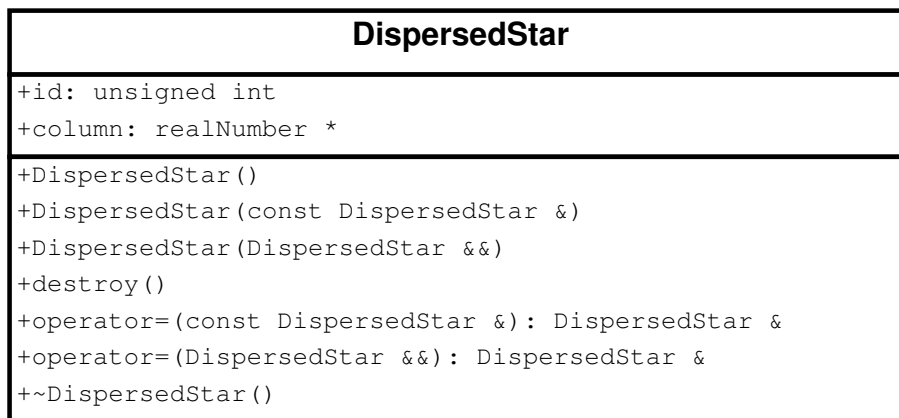


Figura 7.6: Diagrama de la clase **DispersedStar** con notación UML.

Esta clase, ilustrada en la figura 7.6, representa una única estrella dispersada.

La información más relevante de esta clase es la siguiente:

- **id**: valor numérico que indica el número de fila correspondiente a la estrella real asociada.
- **column**: lista de valores numéricos con representación en coma flotante, que contienen los datos transformados o no de aplicar el proceso de dispersión.

Este tipo de dato se trata de un **struct**, al igual que las dos anteriores.

7.3.3.4 Epsilon

Esta clase, ilustrada en la figura 7.7, representa el margen de error permitido en cada índice de búsqueda, también se le conoce como valores épsilon, por su representación matemática con dicha letra griega (ϵ).

Este tipo de dato se trata de un **struct**, al igual que las anteriores.

7.3.3.5 Node

Esta clase representa un nodo de la estructura de árbol AVL. En dicha clase se almacena la siguiente información:

- **value**: es el valor numérico del índice de un determinado nivel de anidamiento. Es utilizado para buscar el conjunto de estrellas artificiales que más se asemejan a una estrella real concreta dentro del margen de error permitido para cada nivel en el proceso de dispersión.
- **content**: o bien es una referencia al siguiente nivel de anidamiento si no se trata del último nivel de anidamiento, o bien si se trata del último nivel de anidamiento es una referencia al conjunto de estrellas artificiales que tienen como campos de búsqueda los mismos valores de los atributos **value** de los niveles anteriores.

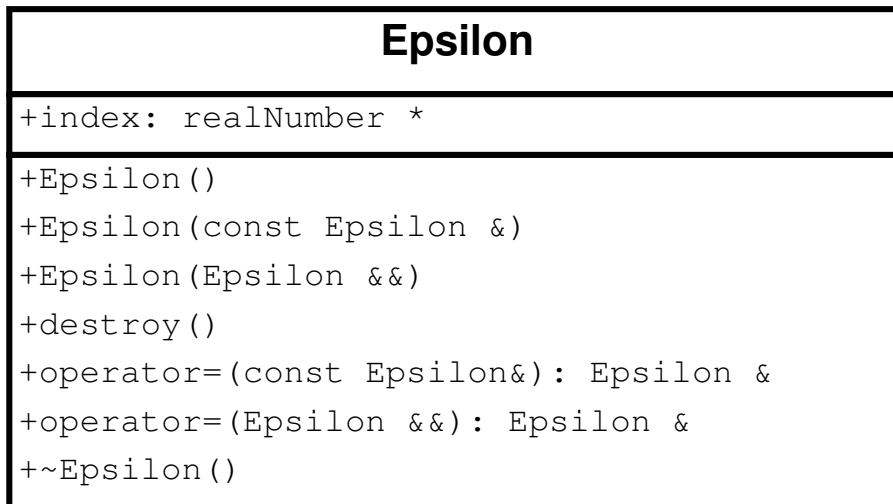


Figura 7.7: Diagrama de la clase Epsilon con notación UML.

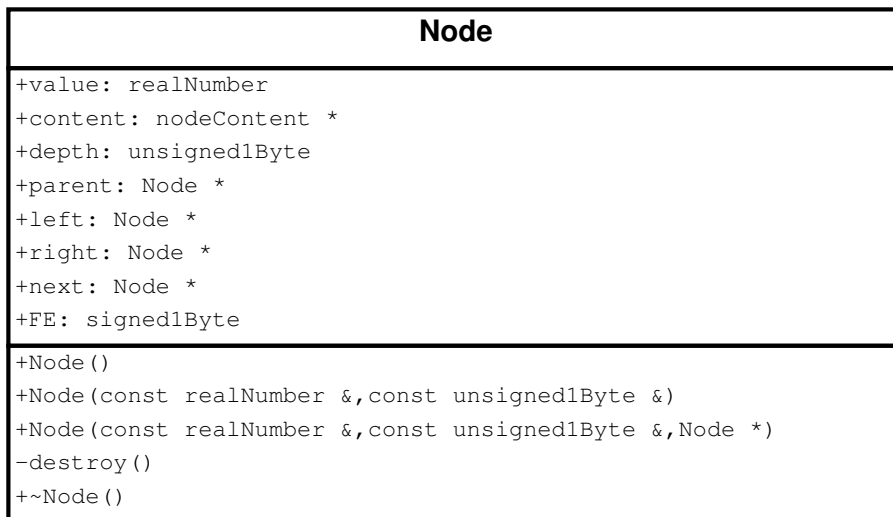


Figura 7.8: Diagrama de la clase Node con notación UML.

- **depth**: es un valor numérico mayor o igual a uno, que indica el nivel de anidamiento. Dicho de otro modo, indica en qué campo de búsqueda, especificado en el fichero de parámetros, se encuentra el programa. Por tanto, este campo determina el tipo de enlace del atributo **content**.
- **parent**: se trata de un enlace hacia el nodo padre del mismo nivel de anidamiento. Al tratarse de una estructura en forma de árbol, el nodo raíz tendrá esta referencia vacía. Es útil para poder navegar por el árbol con la intención de construir el atributo **next**.
- **left**: se trata de una referencia que enlaza con un subárbol cuyos valores del campo **value** son menores al valor ese mismo campo del nodo actual.
- **right**: se trata de una referencia que enlaza con un subárbol cuyos valores del campo **value** son mayores al valor ese mismo campo del nodo actual.
- **next**: se trata de una referencia que enlaza con un nodo cuyo campo **value** es el primero que tiene mayor valor numérico al mismo campo del nodo actual de entre el total de nodos de ese mismo nivel.

Esta referencia es vacía en el proceso de cargar los índices o características leídas desde el fichero de entrada. Un vez finalizada la construcción completa de los índices de búsqueda, se procede a crear las referencias de este campo para todos los nodos.

El nodo con mayor valor tendrá esta referencia vacía.

Este campo será el que facilite la transformación desde una estructura anidada en forma de árbol hacia una estructura anidada en forma de lista.

- **FE**: valor numérico que determina el factor de equilibrio de los subárboles izquierdo y derecho de un nodo en cuestión. Este campo es utilizado para las labores de balanceo del árbol AVL.

Se ilustra en la figura 7.8.

7.3.3.6 Tree

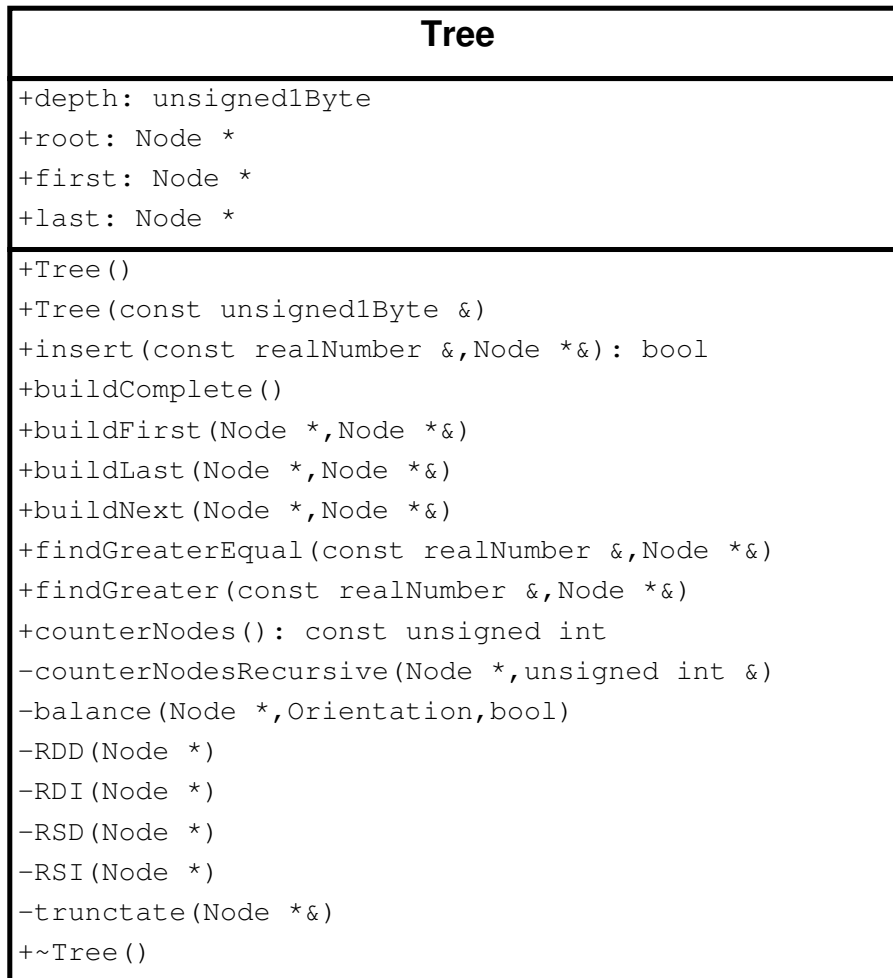


Figura 7.9: Diagrama de la clase **Tree** con notación UML.

Esta clase representa la estructura de árbol AVL que contiene los valores de un único índice o característica de búsqueda.

Esta clase es la encargada de implementar las tareas de inserción y balanceo, propias de esta estructura.

En dicha clase se almacena la siguiente información:

- **depth**: es un valor numérico mayor o igual a uno, que indica el nivel de anidamiento. Dicho de otro modo, indica en qué campo de búsqueda, especificado en el fichero de parámetros, se encuentra el árbol.
- **root**: es una referencia que enlaza con el nodo raíz del árbol. Normalmente el campo **value** del nodo raíz se puede tomar como referencia del valor intermedio entre los valores introducidos.
- **first**: es una referencia que enlaza con el nodo cuyo campo **value** tiene el menor valor de todos los nodos introducidos. Utilizado para transformar la estructura de árbol en la estructura de lista, determinando la condición de inicio del bucle encargado de dicha labor.
- **last**: es una referencia que enlaza con el campo **next** del nodo cuyo campo **value** tiene el mayor valor de todos los nodos introducidos. Utilizado para transformar la estructura de árbol en la estructura de lista, determinando la condición de parada del bucle encargado de dicha labor.

Clase ilustrada en la figura 7.9.

7.3.3.7 Element

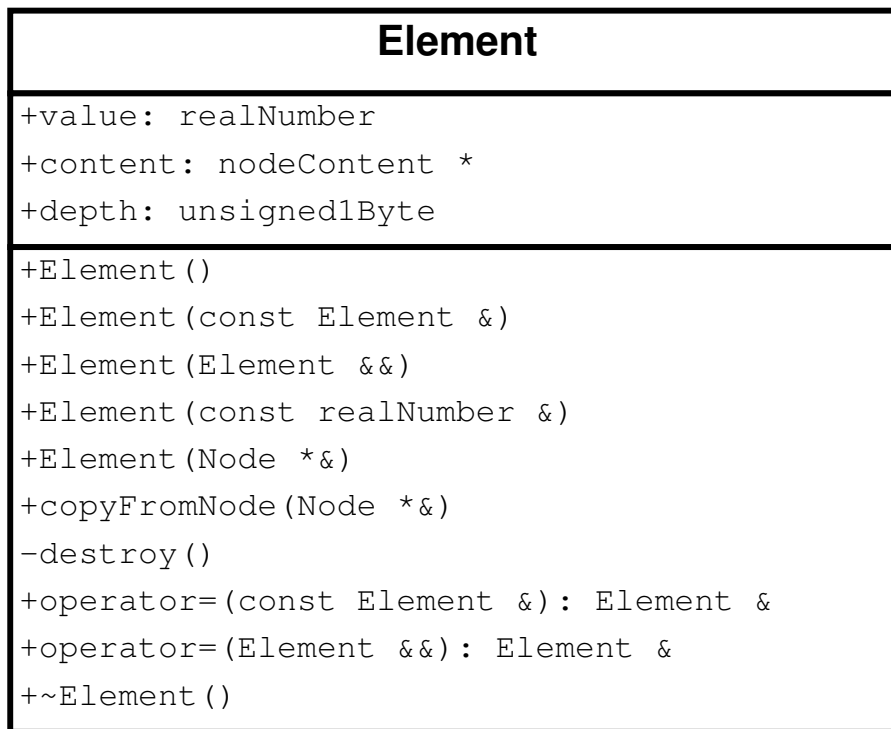


Figura 7.10: Diagrama de la clase **Element** con notación UML.

Esta clase representa un único elemento de la estructura de lista. En dicha clase se almacena la siguiente información:

- **value**: es el valor numérico del índice de un determinado nivel de anidamiento. Es utilizado para buscar el conjunto de estrellas artificiales que más se asemejan a una estrella real concreta dentro del margen de error permitido para cada nivel en el proceso de dispersión.
- **content**: o bien es una referencia al siguiente nivel de anidamiento si no se trata del último nivel de anidamiento, o bien si se trata del último nivel de anidamiento es una referencia al conjunto de estrellas artificiales que tienen como campos de búsqueda los mismos valores de los atributos **value** de los niveles anteriores.
- **depth**: es un valor numérico mayor o igual a uno, que indica el nivel de anidamiento. Dicho de otro modo, indica en qué campo de búsqueda, especificado en el fichero de parámetros, se encuentra el programa. Por tanto, este campo determina el tipo de enlace del atributo **content**.

La clase ilustrada en la figura 7.10.

7.3.3.8 List

Esta clase, ilustrada en la figura 7.11, representa la estructura de lista que contiene los valores de un único índice o característica de búsqueda.

Los datos de interés de esta clase son los siguientes:

- **depth**: es un valor numérico mayor o igual a uno, que indica el nivel de anidamiento. Dicho de otro modo, indica en qué campo de búsqueda, especificado en el fichero de parámetros, se encuentra la lista.
- **elements**: lista de elementos (**Element**) ordenada de menor a mayor en función del valor numérico del campo **value**.

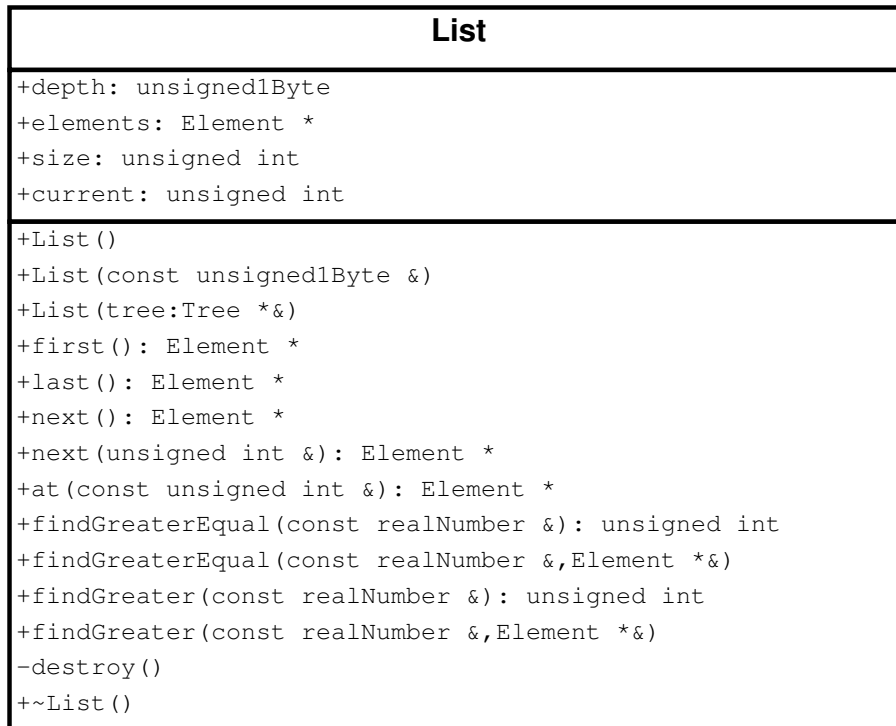


Figura 7.11: Diagrama de la clase `List` con notación UML.

- **size:** valor numérico que indica el tamaño de la lista `elements`.
- **current:** valor numérico que marca el elemento actualmente en uso de la lista `elements`. Su principal uso es dar soporte al método `next`.

7.3.3.9 Index

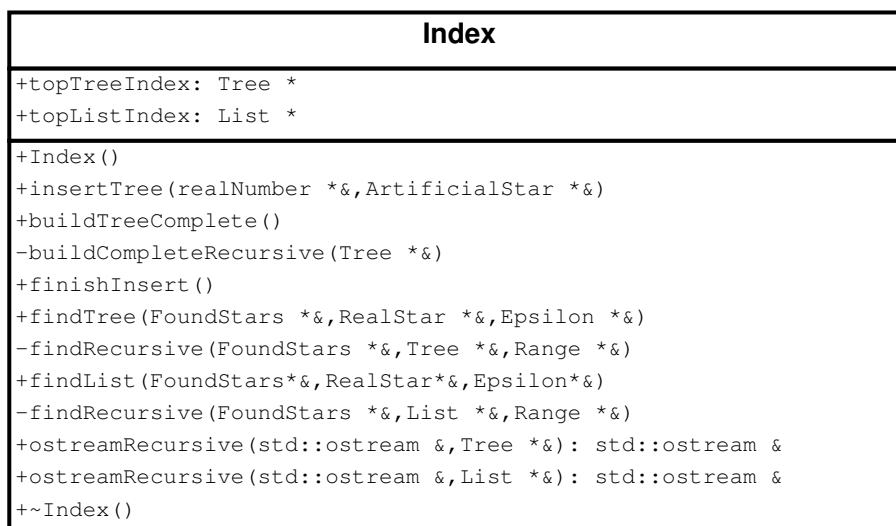


Figura 7.12: Diagrama de la clase `Index` con notación UML.

Esta clase, ilustrada en la figura 7.12, representa la estructura global sobre la que se realizarán las operaciones de búsqueda. Brinda un capa de abstracción que permite realizar operaciones independientemente de la estructura de datos que haya detrás (estructura de árbol y la estructura de lista) y del número de anidamientos que éstas tengan.

Los datos de interés de esta clase son los siguientes:

- **depth**: es un valor numérico mayor o igual a uno, que indica el nivel de anidamiento. Dicho de otro modo, indica en qué campo de búsqueda, especificado en el fichero de parámetros, se encuentra la lista.
- **elements**: lista de elementos (**Element**) ordenada de menor a mayor en función del valor numérico del campo **value**.
- **size**: valor numérico que indica el tamaño de la lista **elements**.
- **current**: valor numérico que marca el elemento actualmente en uso de la lista **elements**. Su principal uso es dar soporte al método **next**.

7.3.4 Funciones destacadas

Es el momento de mencionar las funciones que engloban el resto de funcionalidades desarrolladas en esta implementación. En concreto se trata de cuatro grandes funciones que tienen un cometido específico:

- **checkParameters**: esta función se encarga de leer el fichero de parámetros y comprobar que los datos introducidos son coherentes para no producir una ejecución estéril. Si esto es así, mostrará al usuario los datos leídos y preguntará si está o no conforme antes de proceder con la ejecución.

Si se produjera algún error en el proceso de lectura o comprobación de datos, entonces la ejecución del programa se abortará, mostrando al usuario un mensaje que indica el motivo por el que se abortó la ejecución y generando un fichero de ejemplo que puede servir como fichero de parámetro.

Las posibles causas de que se produzca un error son muy extensas y se pueden consultar en la documentación generada. Algunas de ellas son las siguientes: referencia al fichero de parámetro y a los ficheros de entrada inexistentes, número de argumentos diferente a uno, error de sintaxis en el fichero de parámetros, inexistencia de campos o tipo de datos incorrectos en el fichero de parámetros, errores en las expresiones matemáticas que definen los campos de búsqueda, número mínimo de estrellas requeridas en un conjunto con valor menor que uno, disparidad en la cantidad de elementos de los valores ϵ y los índices de las estrellas reales y artificiales, entre otros.

- **preProcess**: esta función se encarga de realizar las tareas necesarias para dejar todo listo antes de ejecutar el proceso de dispersión.

Las tareas realizadas por esta función son las siguientes: leer las estrellas artificiales del fichero de entrada, crear la estructura anidada de índices de búsqueda y reservar la máxima capacidad de estrellas reales en memoria RAM para procesarlas por lotes.

- **process**: esta función es el que pone en marcha el proceso de dispersión.

Las tareas llevadas a cabo en esta función siguen los siguientes pasos:

1. **datos**: cargar en el lote de memoria reservada para el procesamiento de estrellas reales nuevos los datos desde donde hace referencia el descriptor del fichero madre. Se realiza un análisis léxico y sintáctico si las opciones personalizadas están activas.
2. **expresiones**: calcula las expresiones usadas para el proceso de búsqueda de estrellas artificiales semejantes.
3. **dispersión**: esta tarea se realiza en paralelo para cada estrella real buscando las estrellas artificiales similares, dispersando la si se cumplen las condiciones necesarias y almacenando en el fichero de estrellas dispersadas los resultados.
4. **bucle**: repetir el paso 1, paso 2 y paso 3 hasta que se hayan procesado las estrellas reales indicadas o el descriptor del fichero de estrellas reales marque el final de dicho fichero.
5. **índice**: escribir en un nuevo fichero las estrellas artificiales ordenadas de forma ascendente según los valores que indican las expresiones facilitadas en el fichero de parámetros.

- **postProcess**: esta función se encargará de liberar los recursos solicitados al sistema operativos como memoria RAM, descriptors de ficheros, etc. Este proceso se realiza de forma ordenada según el tipo de recurso.

7.3.5 Recursividad

La sección 7.1 introdujo que en esta implementación se permite definir un número variable de condiciones de búsqueda, mediante la definición de expresiones matemáticas relativa a los datos de las columnas de los ficheros de estrellas reales y artificiales. Estas expresiones se definen en el fichero de parámetros.

La recursividad es la técnica empleada para permitir esta flexibilidad. Por este motivo, la mayoría de las clases descritas en la sección 7.3.3 poseen métodos con nombres similares. Lo normal en estos casos es que uno de ellos se trate de la implementación recursiva y el otro método es el que brinda una capa de abstracción al usuario y prepara el entorno para invocar al método recursivo con los parámetros iniciales para que se ejecute adecuadamente.

Sin embargo, la recursividad puede repercutir negativamente en el rendimiento de esta implementación. Este aspecto negativo, se solventaría desenrollando el bucle recursivo hasta los niveles de anidamiento deseados, pero esta solución no permitiría la flexibilidad de definir un número variable de condiciones de búsqueda.

7.3.6 Analizadores

El análisis de los registros de entrada es un elemento importante en la depuración de errores, cuya causa se deba a elementos externos a esta aplicación. Esto fue introducido en la sección 7.1.

La herramienta externa ANother Tool for Language Recognition (ANTLR), fue desarrollada por Terence Parr⁵. Permite crear analizadores en los lenguajes soportados definiendo una sencilla sintaxis.

Sin embargo, como ya se ha mencionado, el soporte para *C++* se encuentra en una fase experimental para la versión utilizada. Se detectó un *bug* que solamente afecta a la compilación por no declarar de forma adecuada algunas variables. Para solucionarlo, sólo es necesario anteponer la palabra `const` en la declaración de las variables afectadas por este error. La forma de generar el analizador se mencionan en la sección 7.2.3.1.

La utilización del análisis léxico y sintáctico en la ejecución de esta implementación está sujeta a la activación de las opciones personalizadas en el fichero de parámetros.

Si se detecta un error en la fase de análisis, entonces se añadirá una entrada en el fichero de registro indicando el motivo del error.

Existe otro analizador que comprueba si las expresiones matemáticas introducidas en el fichero de parámetros son o no factibles con las expresiones permitidas por la librería ExprTk.

7.3.7 Documentación

Este documento no pretende convertirse en una API para los posibles desarrolladores que en el futuro decidan mantener esta implementación. Si se desea consultar en cada una de las funciones desarrolladas en esta implementación, sus parámetros de entrada, de salida, situaciones asumidas por cada función, etc., se puede consultar la documentación específica en el directorio `/cli/cpu/documentation/crowding-effects`.

La documentación fue generada usando la herramienta Doxygen. De esta forma se garantiza que el código fuente de esta implementación está con las anotaciones necesarias. Esta herramienta genera ficheros en formato HTML, que permite consultar de forma más cómoda las funciones al igual que se puede hacer con cualquier API. Además, relaciona las clases usadas con diagramas.

⁵Parr, *ANother Tool for Language Recognition (ANTLR)*

7.4 Manual de usuario

Esta sección está dedicada a las personas que deseen conocer *grosso modo* la implementación, su funcionamiento, cómo utilizarla, etcétera; es decir, los usuarios. Se remitirá a las secciones donde se explican en mayor detalle para que el usuario las consulte, si es que está interesado en hacerlo.

Esta implementación está orientada a ser ejecutada en un entorno no gráfico o en una interfaz de línea de comandos, también conocida como CLI, por sus siglas en inglés. Es necesario que el usuario disponga de los permisos de ejecución de los ejecutables de esta implementación y de una terminal para ejecutarla. Entre las terminales o *shell* más conocidas están las siguientes: **bash**, **zsh**, **dash**, **csch**, etc.

El usuario debe conocer que existen tres ejecutables en esta implementación: pruebas unitarias, depuración, lanzamiento. Cada una de ellas tiene un cometido específico.

El ejecutable destinado a las pruebas unitarias, se realizó con el siguiente cometido: proporcionar una garantía de que las funciones implementadas se comportan de la forma esperada; servir como ejemplo de invocaciones de las funciones; facilitar la detección de errores en el proceso de desarrollo; dotar a esta implementación de mayor calidad. Para identificar este ejecutable basta con observar el nombre del fichero, llamado **crowding-effects-cli-unit-test**, y se ubica en el directorio **/cli/cpu/Test**. Este ejecutable no recibe parámetros.

El ejecutable destinado a la depuración orientada a los desarrolladores para que estos puedan utilizar herramientas, comúnmente llamadas *debuggers*. Estas utilidades permiten realizar un seguimiento detallado de la ejecución del programa. Para identificar este ejecutable basta con observar el nombre del fichero, llamado **crowding-effects-cli-debug**, y se ubica en el directorio **/cli/cpu/Debug**. Este ejecutable recibe un parámetro, que se comentará más adelante.

El ejecutable de lanzamiento orientada a los usuarios finales de la aplicación. Para identificar este ejecutable basta con observar el nombre del fichero, llamado **crowding-effects-cli-release**, y se ubica en el directorio **/cli/cpu/Release**. Este ejecutable recibe un parámetro, que se comentará más adelante.

A continuación, se describen algunas características orientadas al ejecutable de lanzamiento que los usuarios finales deben tener en cuenta. Estas características también son aplicables al ejecutable destinado a la depuración, pero no son aplicables al ejecutable destinado a las pruebas unitarias.

7.4.1 Funcionamiento

El funcionamiento de esta implementación sigue las siguientes pautas:

- comprobación de los parámetros que evitará algunos errores que el usuario pueda cometer escribiendo el fichero de parámetros.
- solicitud de conformidad con los parámetros leídos.
- creación de la estructura dinámica y transformación en la estructura óptima para el proceso de búsqueda.
- reserva de espacio en memoria RAM para el procesado por lotes de estrellas reales.
- ejecución en paralelo por lotes del proceso de dispersión.
- almacenamiento de los resultados obtenidos del proceso de dispersión.
- almacenamiento del índice de estrellas artificiales según los criterios de ordenación especificados.
- liberación de los recursos solicitados al sistema operativo.

Se puede obtener más información a cerca del funcionamiento en la sección 7.3.4 de este documento.

7.4.2 Parámetros

El algoritmo de la sección 2.2 opera con un conjunto de datos y variables como se puede apreciar. El usuario tiene la potestad de decidir este conjunto de datos y el valor de las variables empleadas. Esta información es nombrada como parámetros de ejecución.

Puesto que la cantidad de parámetros de ejecución es elevada, se decidió incorporar todos los parámetros de ejecución en un único fichero de parámetros.

La ruta del fichero de parámetros debe ser el primer y único argumento que reciba el ejecutable.

El fichero de parámetros, como se ha comentado en ocasiones anteriores, debe seguir el formato JSON. En el glosario de términos se puede encontrar una entrada que dará más información al lector sobre este formato.

Este fichero contiene cuatro bloques que agrupan algunos parámetros de ejecución. El agrupamiento de parámetros de ejecución se realiza atendiendo al conjunto de datos a los que estos hacen referencia. Los bloques son los siguientes:

- **General:** parámetros que no asociados a ninguna de las diferentes clasificaciones de estrellas. Entre estos parámetros se encuentran los siguientes: márgenes de errores, número mínimo que debe tener las estrellas que similes tras el proceso de búsqueda, valor máximo permitido para la dispersión y ruta del fichero de registros.
- **Estrellas artificiales:** parámetros asociados a las estrellas artificiales. Entre estos parámetros están los siguientes: ruta del fichero de estrellas artificiales, número de filas o registro de éstas estrellas que se desea procesar, número de columnas de cada estrella, expresiones matemáticas para crear el índice de búsqueda y ruta del fichero de estrellas ordenadas según el anterior criterio.
- **Estrellas reales:** parámetros asociados a las estrellas reales. Entre estos parámetros están los siguientes: ruta del fichero de estrellas reales, número de filas o registro de éstas estrellas que se desea procesar, número de columnas de cada estrella y expresiones matemáticas para crear el índice de búsqueda.
- **Estrellas dispersadas:** parámetros asociados a las estrellas dispersadas. Entre estos parámetros están los siguientes: ruta del fichero de estrellas dispersadas y número de columnas que debe tener cada estrella (este parámetro debe ser dos unidades superior al indicado en el mismo campo de las estrellas reales).

En el caso de que se produzca algún error en la lectura del fichero de parámetros o no se proporcione una ruta válida, entonces se abortará la ejecución del programa indicando el motivo y generando un fichero de ejemplo del fichero de parámetros. En el fichero de parámetros generado como ejemplo está comentado la utilidad de cada parámetro y el tipo de dato esperado por la aplicación. El usuario puede consultarlo para obtener más información.

Cabe destacar que existe un campo que permite o no activar las opciones personalizadas. Se trata del campo `customized`.

Si adquiere el valor `false`, entonces se asume una ejecución con cuatro niveles de búsqueda, cuyas expresiones matemáticas son `c1`, `c1-c2`, `c5` y `c6` para las estrellas artificiales y reales, los errores permitidos son 0,1, 0,04, 300 y 300 en el mismo orden y, finalmente, no se utilizarán los analizadores. Por tanto se ignora los campos `index` y `epsilons`.

Si adquiere el valor `true`, entonces no se ignoran los campos `index` y `epsilons`, que deberán tener la misma cantidad de elementos, interpretando las expresiones matemáticas una para las estrellas artificiales y otra para las estrellas reales. También se utilizarán los analizadores.

7.4.3 Compilación

Para generar un archivo binario ejecutable a partir del código fuente de esta implementación, el usuario debe satisfacer todos los requisitos de la sección 7.2.1 y utilizar los ficheros Makefile que están localizados en cada directorio de los objetivos de compilación, descritos en la sección 7.2.3.2.

7.4.4 Ejemplo

Un sencillo ejemplo ayudará a comprender mejor lo expuesto hasta ahora sobre esta implementación, la estructuras utilizadas y el resultado esperado para unos parámetros de entrada específicos.

El ejemplo tendrá un conjunto de datos muy limitado y será determinista, ya que el componente de aleatoriedad del proceso de dispersión será forzado para elegir de forma aleatoria un elemento de un conjunto que sólo tiene un elemento.

Se está asumiendo que el directorio actual de la terminal es `/cli/cpu/`, mientras que los ficheros de entrada y salida del programa están `/cli/cpu/files`.

El comando ejecutado en la terminal es el siguiente:

```
./Release/crowding-effects-cli-release ./files/input.json
```

7.4.4.1 Parámetros

Un ejemplo de los parámetros de ejecución se pueden apreciar a continuación:

```

1 {
2   "General"      : {
3     "minSetSize" : 1,
4     "maxSetSize" : 100,
5     "maxValue"   : 200.0,
6     "customized" : true,
7     "epsilons"   : [ 1.0, 1.0, 10.0, 10.0 ],
8     "log"        : "./files/log.txt"
9   },
10  "ArtificialStars" : {
11    "file"       : "./files/artfStar.databs",
12    "rows"      : 3,
13    "columns"   : 6,
14    "index"     : [ "c1", "c1-c2", "c5", "c6" ],
15    "sort_file" : "./files/sort_artfStar.databs"
16  },
17  "RealStars" : {
18    "file"      : "./files/realStar.databs",
19    "rows"     : 1,
20    "columns"  : 6,
21    "index"   : [ "c1", "c1-c2", "c5", "c6" ]
22  },
23  "DispersedStars" : {
24    "file"     : "./files/dispStars.databs",
25    "columns" : 8
26  }
27 }
```

7.4.4.2 Fichero de estrellas artificiales

Se eligió para este ejemplo un conjunto de tres estrellas artificiales. Los datos de dichas estrellas a continuación:

```

1 30 28 33 31 50 80
2 20 18 23 21 60 90
3 10 08 13 11 40 70

```

7.4.4.3 Fichero de estrellas reales

Se eligió para este ejemplo un conjunto de una única estrellas real. Los datos de esta estrella se ilustran a continuación:

```

1 20.5 18.5 500 800 65 85

```

7.4.4.4 Estructuras de datos

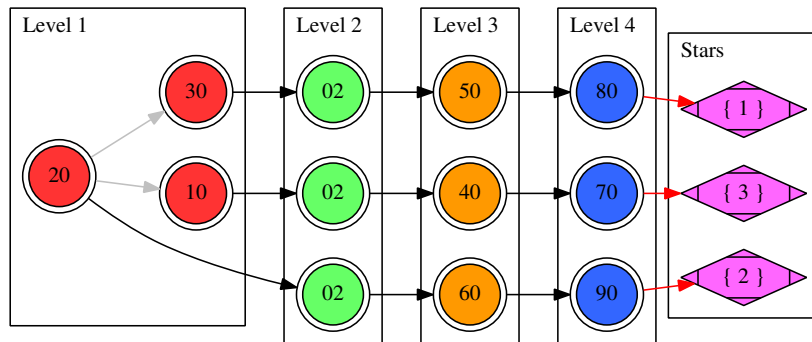


Figura 7.13: Ejemplo aplicado de estructura de árbol de 4 niveles

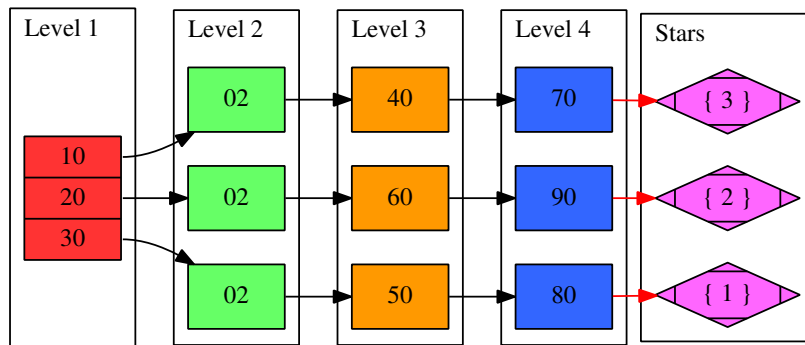


Figura 7.14: Ejemplo aplicado de estructura de lista de 4 niveles

La figura 7.13 muestra el resultado de la creación de la estructura de árbol de cuatro niveles para las estrellas artificiales anteriormente mostradas. Hay que recordar que esta estructura de datos se adapta dinámicamente a nuevas entradas sin mayores esfuerzos.

Una vez finalizada la lectura de todos los registros de estrellas artificiales, se construye la estructura de lista a partir de la estructura de árbol. El resultado es mostrado en la figura 7.14. La estructura en lista ofrece resultados mejores en los tiempos de búsqueda que la estructura en árbol.

7.4.4.5 Búsqueda

Con los datos extraídos de la estrella real, las expresiones matemáticas por defecto y el los valores de los errores por defecto se obtienen los valores mínimo y máximo permitidos realizar las búsquedas en cada uno de los niveles.

En concreto estos valores son los siguientes:

- Nivel 1: el valor mínimo es $19,5 = 20,5 - 1,0$ y el valor máximo es $21,5 = 20,5 + 1,0$.
- Nivel 2: el valor mínimo es $1,0 = 20,5 - 18,5 - 1,0$ y el valor máximo es $3,0 = 20,5 - 18,5 + 1,0$.
- Nivel 3: el valor mínimo es $55,0 = 65,0 - 10,0$ y el valor máximo es $75,0 = 65,0 + 10,0$.
- Nivel 4: el valor mínimo es $75,0 = 85,0 - 10,0$ y el valor máximo es $95,0 = 85,0 + 10,0$.

La única estrella artificial que cumple con los requisitos de búsqueda en todos los niveles es la estrella de la segunda línea del apartado 7.4.4.2.

La selección arbitraria de una estrella del conjunto de estrellas que cumplen las condiciones, en este caso, es determinista, porque sólo es posible elegir una.

7.4.4.6 Dispersión

Seleccionada la estrella artificial que cumple todos los parámetros de búsqueda, es el momento de aplicar el proceso de dispersión.

En esta sección se sigue una estructura de control; es decir, el flujo de ejecución depende del cumplimiento o no de una condición. La condición a evaluar es si el valor los atributos **M1out_j** y **M2out_j** son menores que el valor del campo **maxValue**, también conocido como *tolerancia*.

En este ejemplo, se puede apreciar que:

- **M1out_j**: se corresponde con la columna 3, tiene el valor 23,0 y es menor que 200,0.
- **M2out_j**: se corresponde con la columna 4, tiene el valor 21,0 y es menor que 200,0.

Los valores de la estrella dispersada se obtienen de la única estrella real y de la selección de la única estrella artificial que cumple todas las condiciones de búsqueda.

7.4.4.7 Resultados

Es el momento de escribir los resultado en los ficheros de salida, cuya ruta se indicó en el fichero de parámetros.

El fichero de estrellas dispersadas se aprecia a continuación:

```
1 23.5 21.5 500 800 65 85 1 1
```

El fichero de estrellas artificiales ordenado se ilustra a continuación:

```
1 10 08 13 11 40 70 0
2 20 18 23 21 60 90 1
3 30 28 33 31 50 80 0
```

7.5 Resultados

Los resultados obtenidos de aplicar la implementación recursiva, utilizando la estructura planteada en el apartado 7.3.2 y el paralelismo de la Unidad de Procesamiento Central (CPU) del ordenador, son mostrados en la tabla 7.1, tabla 7.2, tabla 7.3 y tabla 7.4.

La tabla 7.1 muestra el tiempo de ejecución necesario, expresado en segundos, para terminar de procesar el número de estrellas reales del primer par de ficheros, descritos en el apartado 3.1. En esta tabla se han aplicado dos condiciones de búsqueda en el proceso de dispersión.

La tabla 7.2 muestra el tiempo de ejecución necesario, expresado en segundos, para terminar de procesar el número de estrellas reales del primer par de ficheros, descritos en el apartado 3.1. En esta tabla se han aplicado cuatro condiciones de búsqueda en el proceso de dispersión.

Entorno	
• Condiciones	2
• Datos	1º par de ficheros (Ver apartado 3.1)
• Versión	AVL
Estrellas procesadas	OpenMP recursivo
10	0,0042
100	0,0168
1 000	0,1153
10 000	1,057
100 000	8,7514
1 000 000	83,7057
1 500 000	125,827

Tabla 7.1: Resultados de la implementación recursiva en OpenMP con 2 condiciones y conjunto de datos pequeño.

Entorno	
• Condiciones	4
• Datos	1º par de ficheros (Ver apartado 3.1)
• Versión	AVL
Estrellas procesadas	OpenMP recursivo
10	0,1439
100	0,019
1 000	0,1995
10 000	1,7307
100 000	18,7465
1 000 000	226,9398
1 500 000	409,4364

Tabla 7.2: Resultados de la implementación recursiva en OpenMP con 4 condiciones y conjunto de datos pequeño.

La tabla 7.3 muestra el tiempo de ejecución necesario, expresado en segundos, para terminar de procesar el número de estrellas reales del segundo par de ficheros, descritos en el apartado 3.1. En esta tabla se han aplicado dos condiciones de búsqueda en el proceso de dispersión.

Entorno	
• Condiciones	2
• Datos	2º par de ficheros (Ver apartado 3.1)
• Versión	AVL
Estrellas procesadas	OpenMP recursivo
10	0,0315
100	0,0545
1 000	0,4152
10 000	4,1817
100 000	38,2625
1 000 000	383,737
10 000 000	–
16 000 000	–

Tabla 7.3: Resultados de la implementación recursiva en OpenMP con 2 condiciones y conjunto de datos grande.

La tabla 7.4 muestra el tiempo de ejecución necesario, expresado en segundos, para terminar de procesar el número de estrellas reales del segundo par de ficheros, descritos en el apartado 3.1. En esta

tabla se han aplicado cuatro condiciones de búsqueda en el proceso de dispersión.

Entorno	
• Condiciones	4
• Datos	2º par de ficheros (Ver apartado 3.1)
• Versión	AVL
Estrellas procesadas	OpenMP recursivo
10	8,5115
100	0,1615
1 000	1,3283
10 000	11,1537
100 000	96,0288
1 000 000	1 024,29
10 000 000	–
16 000 000	–

Tabla 7.4: Resultados de la implementación recursiva en OpenMP con 4 condiciones y conjunto de datos grande.

Con estos resultados, ya es posible realizar comparativas y calcular la ganancia en velocidad que ofrece esta implementación.

7.6 Ganancia de velocidad

Esta implementación incorpora una estructura, mencionada en el apartado 7.3.2, que en teoría debe ofrecer mejores resultados que la estructura básica.

Sin embargo, como ya se ha mencionado en ocasiones anteriores, existe un factor que puede perjudicar la posible mejora de la estructura planteada. Se trata de la recursividad. La estructura básica se implementó con un algoritmo iterativo, lo que le concede una ventaja frente a esta implementación que contiene un algoritmo recursivo. Es bien conocido, que los algoritmos iterativos, suelen tener mejor rendimiento frente a los algoritmos recursivos.

Es conviene destacar que para realizar las comparaciones se escoga los datos disponibles donde el tamaño del problema a resolver sea mayor. Por eso, se ha elegido los resultados que fueron obtenidos utilizando el segundo par de ficheros. Otro factor a tener en cuenta es el número de condiciones que se deben cumplir en el proceso de búsqueda.

Una vez introducidas las consideraciones previas, es el momento de comparar los resultados:

La tabla 7.5, concentra los resultados de la tabla 4.3 y la tabla 7.3 para facilitar la comparación.

Entorno			
• Condiciones	2		
	2º par de ficheros (Ver apartado 3.1)		
• Datos	$n \cdot m$		AVL
• Versión			
Estrellas procesadas	Secuencial	OpenMP iterativo	OpenMP recursivo
10	0,9245	0,2361	0,0315
100	9,3341	1,9183	0,0545
1 000	93,8433	18,6863	0,4152
10 000	939,9352	186,2368	4,1817
100 000	–	1 859,831	38,2625
1 000 000	–	–	383,737
10 000 000	–	–	–
16 000 000	–	–	–

Tabla 7.5: Resultados de la tabla 4.3 y la tabla 7.3 concentrados.

El gráfico de la figura 7.15 ilustra los tiempos de procesamiento de la tabla 7.5.

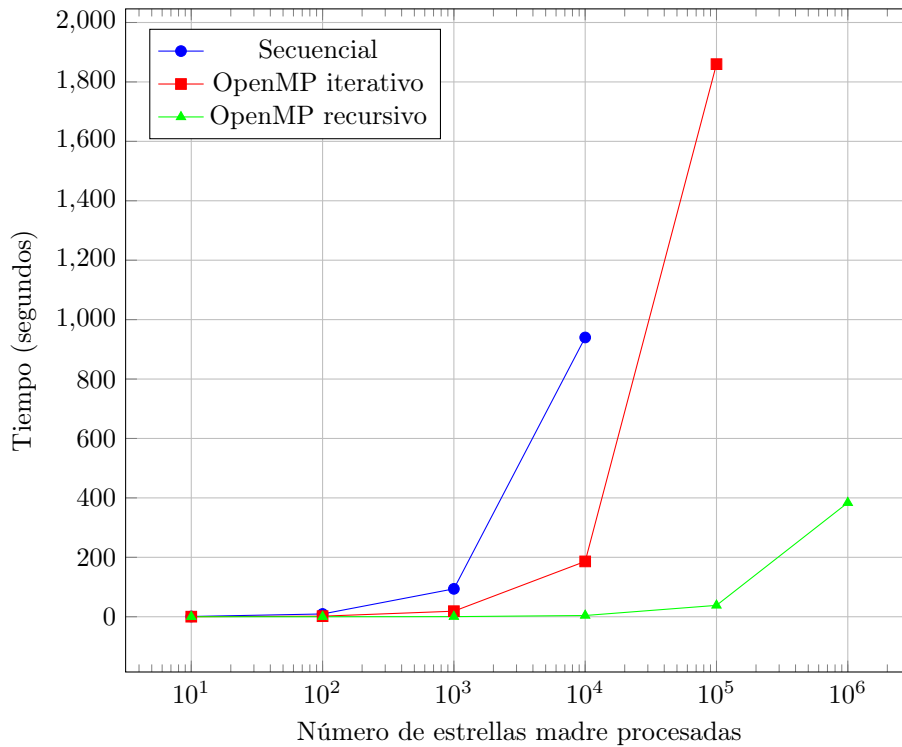


Figura 7.15: Gráfico de tiempos de procesamiento de la tabla 7.5

La tabla 7.6, concentra los resultados de la tabla 4.4 y la tabla 7.4 para facilitar la comparación.

● Condiciones ● Datos ● Versión	Entorno		
	4		
	2º par de ficheros (Ver apartado 3.1)		
	$n \cdot m$		AVL
Estrellas procesadas	Secuencial	OpenMP iterativo	OpenMP recursivo
10	1,1139	0,2796	8,5115
100	11,6606	2,5325	0,1615
1 000	118,5543	23,4513	1,3283
10 000	1 175,1146	231,5997	11,1537
100 000	–	–	96,0288
1 000 000	–	–	1 024,29
10 000 000	–	–	–
16 000 000	–	–	–

Tabla 7.6: Resultados de la tabla 4.4 y la tabla 7.4 concentrados.

El gráfico de la figura 7.16 ilustra los tiempos de procesamiento de la tabla 7.6.

La tabla 7.5 y la tabla 7.6 concentran los resultados de interés de otras tablas; mientras que el gráfico 7.15 y el gráfico 7.16 ilustran sus valores.

A pesar de las desventajas descritas de la implementación recursiva, se puede observar en la tabla 7.5 y la tabla 7.6 que la utilizar la estructura planteada permite superar dichas desventajas con gran holgura.

Se aprecia que el *speedup* que ofrece la implementación recursiva con la estructura planteada en este documento frente a otras implementaciones son las siguientes:

- frente a la implementación secuencial con la estructura básica el *speedup* es de 224,7735 para 10000

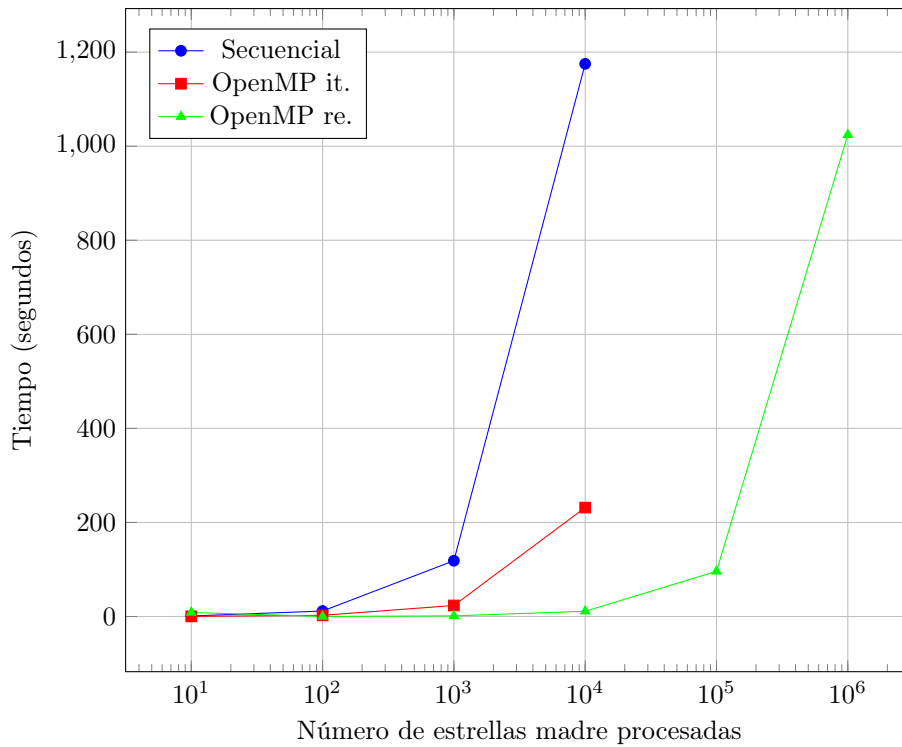


Figura 7.16: Gráfico de tiempos de procesamiento de la tabla 7.6

estrellas procesadas y 2 condiciones.

- frente a la implementación iterativa que utiliza paralelismo de CPU y la estructura básica el *speedup* es de 48,6071 para 100000 estrellas procesadas y 2 condiciones.
- frente a la implementación secuencial con la estructura básica el *speedup* es de 105,3565 para 10000 estrellas procesadas y 4 condiciones.
- frente a la implementación iterativa que utiliza paralelismo de CPU y la estructura básica el *speedup* es de 20,7644 para 100000 estrellas procesadas y 4 condiciones.

Parece que el *speedup* es inversamente proporcional al número de condiciones, aunque no se puede afirmar con rotundidad por no disponer de los conocimientos astrofísicos necesarios para definir más condiciones que tengan coherencia con los datos.

Capítulo 8

Implementación: Paralelismo a nivel de CPU y de GPU

En este apartado explicaremos la segunda implementación que hemos desarrollado para resolver el problema presentado en el apartado 1.1, cuyo pseudocódigo, se muestra en el apartado 2.2. Esta implementación se basa en la estructura de datos propuesta en el capítulo 5 para resolver dicho problema de una forma más eficiente. La implementación en su conjunto se puede descomponer en una serie de etapas. En los apartados posteriores explicaremos cada una de estas etapas.

8.1 Introducción

Para comenzar con la explicación de esta implementación, empecemos por una de las cosas más importantes, el lenguaje de programación utilizado. El lenguaje de programación que se ha usado en esta implementación es **C**. Entre las razones del porqué de esta elección destaca que es un lenguaje que permite un control a bajo nivel lo que lo hace idóneo para realizar implementaciones óptimas. Además haciendo uso de la API del apartado 6.1 y algunas de sus directivas podemos emplear paralelismo a nivel de CPU. Con la plataforma explicada en el apartado 6.2 y sus extensiones podemos usar el paralelismo a nivel de GPU.

Otro de los detalles importantes a tener en cuenta es que esta implementación está **limitada** a realizar **búsquedas y selecciones de estrellas crowding por dos o cuatro condiciones**. Esta es una **diferencia notable** en cuanto a la implementación explicada en el capítulo 7 que **si es capaz** de realizar las búsquedas y selecciones de estrellas crowding por un número arbitrario de condiciones proporcionado como entrada.

El apartado inmediatamente posterior a esta introducción se ha dedicado a describir los requisitos hardware y software necesarios para poder utilizar la presente implementación, así como la instalación del software más crítico. El siguiente apartado describe el formato que ha de tener la entrada (fichero de parámetros). A continuación, explicaremos de forma breve cada una de las etapas en las que se puede subdividir la implementación. Hay un total de seis etapas:

1. Lectura de los ficheros de datos: Fichero Madre y Fichero Crowding de entrada.
2. Cálculo de expresiones: calcular los valores de las características de las estrellas madre y crowding, a partir de los datos leídos en la etapa anterior.
3. Creación de la estructura explicada en el capítulo 5, con los valores de las características correspondientes a las estrellas crowding, que han sido obtenidos en la etapa anterior.
4. Transformación de la estructura de la etapa anterior: Linealización.
5. Realizar el proceso de dispersión, usando la estructura linealizada. A medida que esta etapa avance iremos obteniendo los datos correspondientes a las estrellas dispersas que constituirán el fichero dispersado. También iremos obteniendo los datos necesarios para generar el fichero crowding de salida.

6. Escritura en disco de los resultados obtenidos en la etapa anterior. Fichero dispersado, fichero crowding de salida y fichero de selecciones.

Cada una de las etapas anteriores contará con su propio apartado donde se realizará una explicación algo más detallada.

Posteriormente, dedicaremos un apartado al modo de uso, en el que describiremos la manera de generar los ejecutables (compilación) y la forma de ejecutarlos. Por último, se realizará una prueba de rendimiento donde se mostrarán los resultados de tiempo obtenidos haciendo uso de esta implementación.

8.2 Requisitos: Hardware y software. Instalación del software

En primer lugar, destacar que las personas participantes en el proyecto por parte del IAC nos indicaron que un requisito indispensable era que cualquier implementación desarrollada debería de funcionar en el sistema operativo Linux, dado que es el sistema operativo que se utiliza de forma mayoritaria en el IAC.

El requisito hardware más restrictivo a la hora de usar esta implementación, radica en que si se quiere hacer uso de paralelismo a nivel de GPU, es necesario tener acceso a una máquina que disponga de tarjeta gráfica Nvidia que soporte CUDA. Es posible que en algún momento no dispusiéramos de acceso a una máquina con el mencionado hardware, por tanto, se han realizado **dos versiones** de la misma implementación:

- La versión **OPENMP** que sólo permite paralelismo a nivel de CPU.
- La versión **CUDA-OPENMP** que permite paralelismo a nivel de CPU y de GPU.

La única diferencia notable entre ambas versiones es el fichero de parámetros de entrada, la versión **CUDA-OPENMP** necesita un parámetro adicional (para determinar qué tipo de paralelismo se usa) que la versión **OPENMP** no necesita.

A continuación describiremos los requisitos de software mínimos necesarios (y las versiones usadas durante la etapa de desarrollo) con los que debe contar la máquina en la que vamos a hacer uso de esta implementación, en su versión **CUDA-OPENMP**:

- Sistema operativo Linux (distribuciones Ubuntu, Kubuntu, Fedora o similares). Se han utilizado diferentes versiones de la distribución Kubuntu. La última de ellas ha sido la 15.04.
- Compilador de C (gcc). La versión de gcc usada ha sido la 4.9.2.
- Librería de cálculo de expresiones (libmatheval). La versión de la librería utilizada es la 1.1.11.
- Driver de Nvidia. La versión del driver Nvidia instalado es la 361.28.
- Compilador Nvidia (NVCC). Este compilador se distribuye dentro del CUDA Toolkit que puede ser descargado desde la página de oficial de CUDA en el siguiente link <https://developer.nvidia.com/cuda-toolkit>. La versión del CUDA Toolkit que se ha utilizado es la 7.5 y la versión de NVCC es la 7.5.17.

Para poder hacer uso de la versión OPENMP será suficiente con los **tres primeros requisitos anteriores**.

Para instalar los requisitos de los últimos cuatro puntos anteriores será necesario abrir **una consola con permisos de administrador**. Los comandos y pasos de instalación de software que siguen a continuación se han llevado a cabo utilizando la distribución Linux Kubuntu.

Para instalar el segundo de los requisitos, compilador de C (gcc), basta con usar el siguiente comando:

```
apt-get install gcc
```

La librería de expresiones, libmatheval, se instala mediante el siguiente comando:

```
apt-get install libmatheval-dev
```

La instalación del siguiente requisito es algo más delicada. La mayor parte de las distribuciones Linux, como es el caso de Kubuntu, vienen con un driver de pantalla instalado por defecto que no sirve para

hacer desarrollos con CUDA. El nombre de este driver en Kubuntu es **nouveau**. Por tanto, hay que desinstalar este driver e instalar el driver de Nvidia que puede ser descargado de la página oficial de Nvidia. Los pasos y comandos a seguir son:

1. Abrimos una **consola tty**, para acceder al sistema fuera del entorno gráfico. Para ello pulsar la siguiente combinación de teclas Ctrl+Alt+F1.

2. Hacemos login en el sistema y obtenemos permisos de administrador (root) ejecutando el comando:

```
sudo su
```

3. Detenemos el entorno gráfico haciendo uso del comando:

```
service sddm stop
```

4. Desactivar la carga del driver nouveau mediante los siguientes comandos:

- a) Creamos el fichero blacklist-nouveau.conf con un editor de texto:

```
vim /etc/modprobe.d/blacklist-nouveau.conf
```

- b) Añadimos las siguientes líneas al fichero creado en el paso anterior y salvamos los cambios:

```
blacklist nouveau
blacklist lbm-nouveau
options nouveau modeset=0
alias nouveau off
alias lbm-nouveau off
```

- c) Creamos el fichero nouveau-kms.conf con un editor de texto:

```
vim /etc/modprobe.d/nouveau-kms.conf
```

- d) Añadimos la siguiente línea al fichero creado en el paso anterior y salvamos los cambios:

```
options nouveau modeset=0
```

- e) Finalmente deshabilitamos la carga del driver con el siguiente comando:

```
update-initramfs -u
```

- f) Instalamos la última versión del driver que hemos descargado de la página de descarga de controladores de Nvidia (<http://www.nvidia.es/Download/index.aspx?lang=en>) ejecutando en la consola:

```
sh NVIDIA-Linux-xAA_BB-XXX.XX.run
```

Para la instalación del último requisito descargamos de la página de descargas de CUDA, el CUDA Toolkit en su versión 7.5. Para instalar el CUDA Toolkit usaremos el siguiente comando:

```
sh cuda_7.5.XX_linux.run
```

Después de realizados todos los pasos anteriores, deberíamos contar con un entorno en el que es posible utilizar esta implementación.

8.3 Formato de entrada: Ficheros de Parámetros

A continuación se van a describir los ficheros de parámetros para las versiones **OPENMP** y **CUDA-OPENMP**.

El contenido del fichero de parámetros para la versión **CUDA-OPENMP** debe ser el siguiente:

- 1: Nombre del fichero madre.
- 2: Nombre del fichero crowding de entrada.
- 3: Nombre del fichero dispersado.
- 4: Nombre del fichero crowding de salida.
- 5: Nombre del fichero de selecciones.
- 6: Número de índices por los que se realizan las búsquedas.
- 7: Indicador que determina el tipo de ejecución a realizar (CUDA ó OPENMP).

- 8: Primera característica que será usada en la búsqueda para las estrellas madre.
- 9: Segunda característica que será usada en la búsqueda para las estrellas madre.
- 10: Tercera característica que será usada en la búsqueda para las estrellas madre.
- 11: Cuarta característica que será usada en la búsqueda para las estrellas madre.
- 12: Primera característica que será usada en la búsqueda para las estrellas crowding.
- 13: Segunda característica que será usada en la búsqueda para las estrellas crowding.
- 14: Tercera característica que será usada en la búsqueda para las estrellas crowding.
- 15: Cuarta característica que será usada en la búsqueda para las estrellas crowding.
- 16: Margen de error que será aplicado en la primera característica.
- 17: Margen de error que será aplicado en la segunda característica.
- 18: Margen de error que será aplicado en la tercera característica.
- 19: Margen de error que será aplicado en la cuarta característica.
- 20: Valor de tolerancia.
- 21: Número mínimo de estrellas que deben ser seleccionadas.
- 22: Identificador del dispositivo gráfico (GPU) donde se lanzará la ejecución CUDA.
- 23: Número de estrellas madre que serán procesadas cada vez que se llame al kernel.
- 24: Número de threads por bloque.
- 25: Número de bloques.

El contenido de cada una de las líneas es el siguiente:

- Las **cinco primeras líneas** son cadenas de caracteres donde se especificarán **los nombres o las rutas de los ficheros**.
- La **sexta línea** indica el **número de condiciones** por el que se realizarán las búsquedas y selecciones de estrellas crowding. Su valor puede ser **dos (2)** ó **cuatro (4)**. En el caso de ser dos se tendrán en cuenta las dos primeras líneas de características y las dos primeras líneas de los márgenes de error.
- La **séptima línea** indica el **tipo de ejecución** que se debe realizar. Se trata de un número entero cuyo valor puede ser **1 (ejecución OPENMP, paralelismo a nivel de CPU)** ó **2 (ejecución CUDA, paralelismo a nivel de GPU Nvidia)**.
- Las **siguientes ocho líneas** se corresponden con las **expresiones** que se quieren usar para **cada una de las características de ambos tipos de estrellas**. Se pueden escoger expresiones para las estrellas madres y crowding. Se trata de **expresiones matemáticas** donde intervienen las **columnas de los ficheros de entrada**. Puesto que ambos **ficheros de entrada tienen seis columnas**, la forma de denotar a la **columna 1** será mediante la cadena **“c1”** o **“C1”** y así sucesivamente. Los tipos de expresiones que se permiten son **sumas (“+”)**, **restas (“-”)**, **multiplicaciones (“*”)**, **divisiones (“/”)**, **constantes enteras** y **paréntesis** para construir **expresiones de mayor complejidad**.
- Las **siguientes cuatro líneas** se corresponden con el **margen de error** que será aplicado a **cada una de las cuatro características** de las estrellas madres en la búsqueda y selección de estrellas crowding. El **primer margen** le será aplicado a la **primera característica de las estrellas madres** y así sucesivamente. El formato válido serán números decimales con punto (“.”) para separar la parte entera de la real. También se podrá hacer uso de la notación exponencial.
- La **línea 20** se corresponde con el **valor de tolerancia**. Este **parámetro** será usado en el **proceso de dispersión**. El formato válido será el de un número decimal donde se utilizará el carácter punto (“.”) para separar parte entera y real.
- La **línea 21** se corresponde con el **número mínimo de estrellas crowding que se debe seleccionar para que una estrella madre sea dispersada**. Su formato válido será el de un número entero.
- La **línea 22** indica el **dispositivo gráfico que se va usar para la ejecución CUDA**. Se trata de un número entero a partir de 0. En un ordenador de escritorio (PC) el dispositivo 0 suele ser el dispositivo que está usando para los gráficos primarios del PC, por tanto, la ejecución con dicho dispositivo podría llevar a que la pantalla se congele mientras dure la ejecución.

- La **línea 23** indica el **número de estrellas madres que serán procesadas a la vez** en una ejecución del kernel. El tamaño del fichero madre es bastante significativo, por tanto, para procesarlo este tiene que ser dividido en trozos.
- La **últimas dos líneas** se corresponden con **parámetros** que se usan para **configurar la ejecución del kernel CUDA**. En esta configuración se tiene en cuenta el tipo de GPU del que dispone la máquina.

El contenido del fichero de parámetros para la versión **OPENMP** debe ser el siguiente:

- 1: Nombre del fichero madre.
- 2: Nombre del fichero crowding de entrada.
- 3: Nombre del fichero dispersado.
- 4: Nombre del fichero crowding de salida.
- 5: Nombre del fichero de selecciones.
- 6: Número de índices por los que se realizan las búsquedas.
- 7: Primera característica que será usada en la búsqueda para las estrellas madre.
- 8: Segunda característica que será usada en la búsqueda para las estrellas madre.
- 9: Tercera característica que será usada en la búsqueda para las estrellas madre.
- 10: Cuarta característica que será usada en la búsqueda para las estrellas madre.
- 11: Primera característica que será usada en la búsqueda para las estrellas crowding.
- 12: Segunda característica que será usada en la búsqueda para las estrellas crowding.
- 13: Tercera característica que será usada en la búsqueda para las estrellas crowding.
- 14: Cuarta característica que será usada en la búsqueda para las estrellas crowding.
- 15: Margen de error que será aplicado en la primera característica.
- 16: Margen de error que será aplicado en la segunda característica.
- 17: Margen de error que será aplicado en la tercera característica.
- 18: Margen de error que será aplicado en la cuarta característica.
- 19: Valor de tolerancia.
- 20: Número mínimo de estrellas que deben ser seleccionadas.
- 21: Identificador del dispositivo gráfico (GPU) donde se lanzará la ejecución CUDA.
- 22: Número de estrellas madre que serán procesadas cada vez que se llame al kernel.
- 23: Número de threads por bloque.
- 24: Número de bloques.

Este fichero **no tiene la línea 7** que se usa en el fichero de parámetros de la versión **CUDA-OPENMP** para distinguir el tipo de ejecución a realizar.

La cabecera de la función que se encarga de la lectura del fichero de parámetros es:

```
ParametrosEjecucion leerParametrosEntrada(int nparam, char *argv[]);
```

Si el fichero de parámetros cumple con los formatos anteriormente descritos, todos los parámetros se devuelven en una estructura (`ParametrosEjecucion`) que contiene un indicador (`codSalida`) que señala si la lectura fue satisfactoria o no. Si la lectura es satisfactoria el valor de este indicador será "1" y si no lo es un valor menor que "0".

Esta función está definida en `parametros_entrada.c`, siendo su fichero de cabecera `parametros_entrada.h`. La estructura `ParametrosEjecucion` está definida en el fichero `tipos.h`.

8.4 Almacenamiento de datos

En este apartado se explicarán las estructuras de datos usadas para almacenar los datos de entrada, los datos intermedios y los datos de salida.

Recordemos que esta versión está orientada a resolver el problema usando paralelismo a nivel de GPU mediante CUDA. Por tanto, la mejor forma para el almacenamiento de datos consiste en hacer el menor número de reservas posibles del mayor tamaño de posible. De esta forma, la estructura utilizada para el almacenamiento son vectores.

A continuación se describirán los tipos de datos utilizados para almacenar la información relativa a las estrellas:

- La siguiente definición se utiliza para los valores flotantes de las estrellas.

```
typedef double tipoDato
```

- La siguiente definición se usa para guardar los datos correspondientes a las **estrellas crowding de entrada**.

```
typedef tipoDato* ListaDatosCrowding
```

- La siguiente definición se utiliza para guardar los datos correspondientes a las **estrellas madre**.

```
typedef tipoDato* ListaDatosMadre
```

- La siguiente definición se usa para guardar los **valores de las expresiones** de las **estrellas crowding de entrada**.

```
typedef tipoDato* ListaDatosCrowdingExp
```

- La siguiente definición se utiliza para guardar los **valores de las expresiones** de las **estrellas madre**.

```
typedef tipoDato* ListaDatosMadreExp
```

- La siguiente definición almacena **ciertos valores** de cada una de las estrellas crowding de entrada. Volviendo al pseudocódigo del apartado 2.2, más concretamente en las líneas 29 y 47, se selecciona una estrella crowding de forma aleatoria de entre todas las que fueron devueltas por los procesos de búsqueda. Si nos fijamos desde esa línea en adelante, de la estrella crowding seleccionada se utilizan **cuatro** (**M1in**, **M2in**, **M1in**, **M2out**) de los **seis** valores que componen este tipo de estrellas.

```
typedef tipoDato* ListaDatosCrowdingParcial
```

- La siguiente definición almacenará los dos indicadores enteros del fichero dispersado explicados en el apartado 2.2 y otro más. Este último indicador (**novena columna del fichero dispersado**) se utiliza para saber si se desbordó el conjunto de estrellas seleccionadas, su valor será 1 si hubo desbordamiento y 0 en caso contrario.

```
typedef int* ListaDatosDispersoEntero
```

- Una variable con el siguiente tipo guardará los datos reales correspondientes a las estrellas dispersadas.

```
typedef tipoDato* ListaDatosDispersoReal
```

- Una variable con el siguiente tipo almacenará un entero para cada estrella crowding que indicará el número de veces que se ha usado la estrella en el proceso de dispersión.

```
typedef int* ListaUsoEstrellasCrowding
```

Los siguientes tipos de datos son la base de la estructura explicada en el capítulo 5.

- Una variable con el siguiente tipo se corresponde con un **nodo rojo** de la figura 5.1 del capítulo 5. Recordemos que estos nodos almacenarán un identificador de estrella crowding (campo **id**) y un enlace a otro nodo del mismo tipo.

```
typedef nodoEspecial* ListaNodosEspeciales
```

- Una variable con el siguiente tipo se corresponde con un **nodo negro** de la figura 5.1 del capítulo 5. Recordemos que estos nodos almacenan los valores de las características (campo **dato**). Este tipo de nodo resulta ser un nodo de un árbol AVL, por tanto, incluye otros dos enlaces (campo **izquierdo** y **derecho**) a nodos del mismo tipo, un enlace al nodo padre al que pertenece (campo **padre**) y el factor de equilibrio (campo **FE**). Finalmente, el último campo (**siguiente_nivel**) se corresponde con los **enlaces naranjas** entre niveles que se muestran en la figura 5.1 del capítulo 5.

```
typedef tipoNodo* Arbol
```

El siguiente tipo constituye un **tipo básico** que se utilizará **para almacenar la transformación de la estructura del capítulo 5** en una **estructura lineal** que será la que utilicemos en el proceso de búsqueda y selección tanto en ejecuciones CUDA y OPENMP. Los campos que componen este tipo

serán explicados en el apartado 8.8 cuando se explique la transformación de la estructura. Es muy importante señalar que **ambas estructuras**, son **equivalentes**, es decir, contienen la misma información. La necesidad de esta estructura se debe al uso de **CUDA**.

```
typedef tipoIndicePlano* ListaIndicePlano
```

Todos los tipos descritos anteriormente se encuentran definidos en el fichero `tipos.h`.

Las siguientes definiciones se corresponden con las estructuras que permitirán la resolución del problema en base a dos o cuatro características.

- Las definiciones que se muestran a continuación se utilizarán para almacenar la estructura que se muestra en la figura 5.1 del capítulo 5. **Los niveles 3 y 4 mostrados en la figura 5.1 no existen en IndiceAVL2**. Los campos y su significado serán explicados en el apartado de creación de la estructura (8.7).

```
typedef tipoIndiceAVL2* IndiceAVL2
typedef tipoIndiceAVL4* IndiceAVL4
```

- Las definiciones que se muestran a continuación se usarán para almacenar las transformaciones de las estructuras anteriores. `IndiceVector2` almacenará la transformación de `IndiceAVL2` e `IndiceVector4` la de `IndiceAVL4`. Los campos y su significado serán explicados en el apartado en el que se realiza la transformación de la estructura (8.8).

```
typedef tipoIndiceVector2* IndiceVector2
typedef tipoIndiceVector4* IndiceVector4
```

Las definiciones de `IndiceAVL2` e `IndiceVector2` se encuentran en `tipos2.h`. Las definiciones de `IndiceAVL4` e `IndiceVector4` se encuentran en `tipos4.h`.

8.5 Lectura de ficheros de datos

En este apartado se explica el proceso de lectura de los ficheros de entrada: fichero madre y fichero crowding de entrada.

El proceso de lectura se compone de los siguientes pasos, para ambos ficheros:

- Un primer paso en el que se recorre el fichero para determinar el número de líneas correctas. Las líneas que no son correctas se descartan y se informa al usuario mediante un mensaje indicando el número de línea. Recordemos que cada línea correcta contiene la información asociada a una estrella.
- Una vez tenemos el número de líneas correctas, se realiza una reserva de memoria.
- A continuación se vuelve a leer el fichero y se almacenan las líneas correctas en el área de memoria reservada en el paso anterior.
- Finalmente el proceso devuelve el área reservada con los datos leídos.

Para realizar las reservas de memoria hay una serie de constantes, definidas en el fichero `constantes.h`, que indican el número de datos que componen cualquiera de los dos tipos de estrellas.

- Para las estrellas madre la constante es:

```
NUMERO_COLUMNAS_MADRE
```

- Para las estrellas crowding la constante es:

```
NUMERO_COLUMNAS_CROWDING
```

Para poder acceder a los distintos valores de las estrellas, se han definido una serie de constantes en el fichero `constantes.h`.

- Para las estrellas madre son:

```
COL_MADRE_M1
COL_MADRE_M2
COL_MADRE_EDAD
COL_MADRE_METALICIDAD
COL_MADRE_X
COL_MADRE_Y
```

- Para las estrellas crowding son:

```
COL_CROWDING_M1I
COL_CROWDING_M2I
COL_CROWDING_M1O
COL_CROWDING_M2O
COL_CROWDING_X
COL_CROWDING_Y
```

Por ejemplo, para acceder al valor de metalicidad de una **estrella madre** en `listaDatosMadre` haremos lo siguiente:

```
listaDatosMadre[i * NUMERO_COLUMNAS_MADRE + COL_MADRE_METALICIDAD]
// siendo i el número de estrella madre
```

Las cabeceras de las funciones que se encargan de leer los ficheros de datos son las siguientes:

- Para las estrellas madre:

```
ListaDatosMadre LeerFicheroMadre_n
(
    ParametrosEjecucion *parametrosEjecucion
);
```

- Para las estrellas crowding:

```
ListaDatosCrowding LeerFicheroCrowding_n
(
    ParametrosEjecucion *parametrosEjecucion
);
```

La definición de estas dos funciones se puede encontrar en el fichero `crowding_madre.c` y el correspondiente fichero de cabecera es `crowding_madre.h`.

8.6 Cálculo de expresiones. Características madre y crowding

En este apartado se explicará el proceso que debemos seguir para realizar la evaluación de las expresiones especificadas en el fichero de parámetros para cada una de las estrellas. Para llevar a cabo dicha evaluación nos apoyaremos en una librería.

Antes de comenzar con la explicación del proceso, conviene señalar:

- La librería utilizada para llevar a cabo la evaluación de expresiones es, la **librería GNU libmatheval**.
- Los campos `exps_madre` y `exps_crowding` de la estructura `ParametrosEjecucion` contienen las cadenas de caracteres para las expresiones. El campo `exps_madre` contiene las expresiones de las estrellas madre. El campo `exps_crowding` contiene las expresiones de las estrellas crowding.
- Antes de ejecutar cualquiera de las funciones que calculan las expresiones se ha realizado una reserva de memoria para almacenar los datos que se van a generar. El tamaño del área de memoria a reservar depende del número de estrellas y del valor del campo `numIndices` de `ParametrosEjecucion`. El valor de dicho campo puede ser **2** o **4**.

- Recordar que las expresiones están formadas por los nombres de columnas que contienen los ficheros madre y crowding de entrada. Ambos ficheros tienen seis columnas, por tanto el formato válido para denotar la columna 1 en las expresiones será mediante la cadena “c1” o “C1” y así sucesivamente para las columnas restantes.

Los pasos que hemos seguido para evaluar una expresión, usando la librería son:

1. Creamos el evaluador asociado (`f_expr`) a la expresión usando la función de librería `evaluator_create`.
2. Llamamos a la función `evaluator_get_variables` con el evaluador `f_expr`, con esto obtendremos el **número de columnas** (`numCols`) y los **nombres de las columnas** (`nombresColumnas`) que aparecen en la expresión. Convertimos los **nombres de las columnas** en **índices numéricos** para acceder a los datos necesarios para poder evaluar la expresión para cada estrella. Los datos están almacenados en `listaDatosCrowding` y `listaDatosMadre`.
3. Ahora iteramos sobre las estrellas y hacemos uso de los **índices numéricos** obtenidos para copiar los datos en un **vector auxiliar** (`valores`) de tamaño `numCols`. Llamamos a la función `evaluator_evaluate` (pasándole `f_expr`, `numCols`, `nombresColumnas` y `valores`) para evaluar el valor de la expresión para cada estrella. Los valores que va devolviendo esta función se van almacenando en una **matriz auxiliar**.

Repetiremos el proceso anterior con cada expresión. Hemos iterado sobre las expresiones y luego sobre las estrellas, por tanto, las dimensiones de la **matriz auxiliar** tendrá tantas filas como expresiones y tantas columnas como estrellas. Para una mayor simplicidad en las operaciones posteriores, se ha decidido transponer esta matriz. El resultado de la transposición será almacenado en `listaDatosCrowdingExp` y en `listaDatosMadreExp` respectivamente.

Las reservas de memoria para `listaDatosCrowdingExp` y `listaDatosMadreExp` se han realizado antes de llamar a las funciones de cálculo de expresiones. Para realizar estas reservas se ha utilizado el campo `numIndices` de la estructura `ParametrosEjecucion`.

Para poder acceder a los valores de las expresiones hay una serie de constantes definidas en `constantes.h`. Las constantes son:

```
EXP_DATO_1
EXP_DATO_2
EXP_DATO_3
EXP_DATO_4
```

Por ejemplo, para acceder al valor de la segunda expresión de una **estrella crowding** en `listaDatosCrowdingExp` haremos lo siguiente:

```
listaDatosCrowdingExp[i * parametrosEjecucion.numIndices + EXP_DATO_2]
// siendo i el número de estrella crowding
```

Las cabeceras de las funciones que realizan el cálculo de expresiones son las siguientes:

- Para las estrellas madre:

```
void calculoExprMadre
(
    ListaDatosMadre listaDatosMadre,
    ParametrosEjecucion parametrosEjecucion,
    ListaDatosMadreExp listaDatosMadreExp
);
```

- Para las estrellas crowding:

```
void calculoExprCrowding
(
    ListaDatosCrowding listaDatosCrowding,
    ParametrosEjecucion parametrosEjecucion,
    ListaDatosCrowdingExp listaDatosCrowdingExp
);
```

La definición de estas dos funciones se encuentra en el fichero `calculo_expresiones.c` y su correspondiente fichero de cabecera es `calculo_expresiones.h`.

En la pareja de ficheros anteriores también hay definida otra función, su cabecera es la siguiente:

```
void copiarDatosCrowdingParcial
(
    ListaDatosCrowding listaDatosCrowding,
    ParametrosEjecucion parametrosEjecucion,
    ListaDatosCrowdingParcial listaDatosCrowdingParcial
);
```

En el proceso de dispersión sólo se necesitan **cuatro** (**M1in**, **M2in**, **M1out**, **M2out**) de los **seis** valores de las estrellas crowding. La función anterior se encarga de realizar la copia de esos cuatro datos desde `listaDatosCrowding` a `listaDatosCrowdingParcial`.

La reserva de memoria para almacenar la copia de estos valores se ha realizado antes de llamar a la función. El número de datos necesarios para cada estrella crowding se ha definido mediante una constante `constantes.h`. El nombre de la constante es:

```
NUMERO_COLUMNAS_CROWDING_PARCIAL
```

Las constantes definidas para acceder a cada uno de esos cuatro datos son:

```
COL_CROWDING_PARCIAL_M1I
COL_CROWDING_PARCIAL_M2I
COL_CROWDING_PARCIAL_M1O
COL_CROWDING_PARCIAL_M2O
```

8.7 Creación de la estructura del capítulo 5: Índice AVL

Una vez evaluadas las expresiones para las estrellas crowding podemos proceder a construir la estructura del capítulo 5.

Recordemos que esta implementación resuelve el problema solamente para dos o cuatro índices por lo que utilizaremos tipos de datos y funciones específicas dependiendo del número de índices. El número de índices a utilizar se especifica en el fichero de parámetros y se guarda en el campo `numIndices` de la estructura `ParametrosEjecucion`.

La estructura utilizada está basada en árboles AVL. Para crear la estructura es necesario utilizar la función de inserción en árboles AVL y las funciones auxiliares que ella utiliza.

- La función de inserción inserta los datos reales en los árboles AVL de los distintos niveles (Nivel 1 hasta Nivel 4) de la estructura. La función devuelve un puntero al nodo insertado o encontrado (en caso de que no haya sido necesario crear un nuevo nodo porque el dato que se está insertando ya existe). La función actualiza un **contador** que indica el número de nodos que hay en un nivel determinado. Una vez que se han insertado todos los datos reales de la estrella crowding en la estructura, se insertará en el nivel de IDs su identificador. La estructura usada en el nivel de IDs es una lista y el tipo utilizado es `ListaNodosEspeciales`. La cabecera de la función de inserción en árboles AVL es:

```
pNodo auxCrearIndiceAVL
(
    Arbol *a,
    tipoDato dat,
    int *contador
);
```

- Las funciones auxiliares utilizadas por la función anterior realizan las **operaciones necesarias para mantener la condición de equilibrio** que deben **cumplir** los **árboles AVL**. La implementación de estas funciones se ha tomado de la página C++ con Clase, cuya URL es <http://>

//c.conclase.net/. Las cabeceras de estas funciones son:

```
void Equilibrar(Arbol *a, pNodo nodo, int rama, int nuevo);
void RSI(Arbol *raiz, pNodo nodo);
void RSD(Arbol *raiz, pNodo nodo);
void RDI(Arbol *raiz, pNodo nodo);
void RDD(Arbol *raiz, pNodo nodo);
```

Como ya hemos dicho, esta implementación puede realizar la búsqueda y selección de estrellas crowding en base a dos o cuatro características únicamente. Para ello y de forma temporal se crea la estructura explicada en el capítulo 5.

- La función `CrearIndiceAVL_2` crea una estructura muy similar a la mostrada en la figura 5.1. La única diferencia es que los niveles 3 y 4 de esa figura, **no existen** en la estructura creada por esta función y los enlaces naranjas del nivel 2 están conectados con los nodos rojos del nivel de IDs. Esta función está definida en `indice_avl2.c`. La cabecera de la función es:

```
IndiceAVL2 CrearIndiceAVL_2
(
    ListaDatosCrowdingExp list,
    ParametrosEjecucion parametrosEjecucion
);
```

- La función `CrearIndiceAVL_4` crea la misma estructura que la mostrada en la figura 5.1. Esta función está definida en `indice_avl4.c`. La cabecera de la función es:

```
IndiceAVL4 CrearIndiceAVL_4
(
    ListaDatosCrowdingExp list,
    ParametrosEjecucion parametrosEjecucion
);
```

- Estas funciones reciben como entrada los datos correspondientes a la evaluación de expresiones de las estrellas crowding y devuelven la estructura. Para construir la estructura realizan el proceso descrito en el capítulo 5.

Para almacenar las estructuras anteriores se han definido dos tipos.

- El tipo `IndiceAVL2` está definido en el fichero `tipos2.h` y contiene los siguientes campos:

- **Puntero al nodo raíz** de la estructura:

```
Arbol indice;
```

- **Tres contadores** que indican el **número de nodos que hay en cada nivel**:

```
int numeroNodosNivel1;
int numeroNodosNivel2;
int numeroEstrellasIndice;
```

- El tipo `IndiceAVL4` está definido en el fichero `tipos4.h` y contiene los siguientes campos:

- **Puntero al nodo raíz** de la estructura:

```
Arbol indice;
```

- **Cinco contadores** que indican el **número de nodos que hay en cada nivel**:

```
int numeroNodosNivel1;
int numeroNodosNivel2;
int numeroNodosNivel3;
int numeroNodosNivel4;
int numeroEstrellasIndice;
```

- Los **contadores** son **muy importantes** ya que se **utilizarán** en la **transformación** que explicaremos en el siguiente apartado.

También se han implementado funciones para liberar la memoria reservada para **índice AVL**.

- La función principal que libera la memoria de `IndiceAVL2` está definida en `indice_avl2.c` y su cabecera es:

```
void PodarIndiceAVL_2
(
    IndiceAVL2 *indiceA
);
```

- La función principal que libera la memoria de `IndiceAVL4` está definida en `indice_avl4.c` y su cabecera es:

```
void PodarIndiceAVL_4
(
    IndiceAVL4 *indiceA
);
```

8.8 Linealización de la estructura Índice AVL: Índice Vector

Una vez generado el índice AVL, el siguiente paso consiste en transformar esa estructura en una estructura lineal, **índice vector**. Ambas estructuras son equivalentes y contienen la misma información. La estructura índice vector es necesaria debido al uso de paralelismo a nivel de GPU con CUDA.

La estructura índice vector se construye **recorriendo la estructura índice AVL** de la **misma manera** que recorremos un **árbol AVL en inorden**.

Supongamos que tenemos un índice AVL de cuatro niveles como el que se muestra en la figura 5.1. Los cuatro primeros niveles (Nivel 1 hasta Nivel 4) serán representados mediante vectores cuyo tipo básico será `ListaIndicePlano`. El nivel de IDs será representado mediante un vector de enteros y almacenará los identificadores de estrellas crowding. Para reservar memoria para estos **cinco** vectores se utilizan los **contadores** correspondientes a cada nivel que fueron explicados en el apartado anterior. Estos cinco vectores y los contadores constituyen el tipo `IndiceVector4` definido en `tipos4.h`. El tipo `IndiceVector2` definido en `tipos2.h` ha sido definido para un índice AVL de dos niveles. El tipo `ListaIndicePlano` está definido en `tipos.h`.

El tipo básico `ListaIndicePlano` es una estructura que contiene tres campos. Este tipo de dato sirve para representar los **nodos negros** de la estructura de la figura 5.1.

- El campo `dato` almacenará el valor real de un nodo negro de la figura 5.1.

```
tipoDato dato
```

- Los campos `lower` y `upper` son dos enteros que sirven para indicar el **rango** en el que se encuentran en el vector del siguiente nivel **los valores reales asociados al campo dato** o **los identificadores de estrellas crowding** (si nos encontramos en el último nivel con nodos negros). El campo `lower` indica el índice inferior y el campo `upper` el superior.

```
int lower, upper
```

Las funciones que realizan la transformación de **índice AVL** a **índice vector** son las siguientes:

- Para la transformación de `IndiceAVL2` a `IndiceVector2` tenemos:
 - La función principal `CrearIndiceVector_2` transforma la estructura `IndiceAVL2` a `IndiceVector2`. Su cabecera es:

```
IndiceVector2 CrearIndiceVector_2(IndiceAVL2 indiceA);
```

- Para realizar la transformación, la función `CrearIndiceVector_2` utiliza tres funciones auxiliares. Estas funciones auxiliares recorren cada uno de los niveles de `IndiceAVL2` a medida que

se va realizando la transformación. Las cabeceras de estas funciones son:

```
void auxCrearNivelLista_2
(
    ListaNodosEspeciales lista,
    IndiceVector2 indiceV
);
void auxCrearNivel2_2(Arbol a, IndiceVector2 indiceV);
void auxCrearNivel1_2(Arbol a, IndiceVector2 indiceV);
```

- Estas funciones están definidas en `indice_vector2.c`.

- Para la transformación de `IndiceAVL4` a `IndiceVector4` tenemos:

- La función principal `CrearIndiceVector_4` transforma la estructura `IndiceAVL4` a `IndiceVector4`. Su cabecera es:

```
IndiceVector4 CrearIndiceVector_4(IndiceAVL4 indiceA);
```

- Para realizar la transformación, la función `CrearIndiceVector_4` utiliza cinco funciones auxiliares. Estas funciones auxiliares recorren cada uno de los niveles de `IndiceAVL4` a medida que se va realizando la transformación. Las cabeceras de estas funciones son:

```
void auxCrearNivelLista_4
(
    ListaNodosEspeciales lista,
    IndiceVector4 indiceV
);
void auxCrearNivel4_4(Arbol a, IndiceVector4 indiceV);
void auxCrearNivel3_4(Arbol a, IndiceVector4 indiceV);
void auxCrearNivel2_4(Arbol a, IndiceVector4 indiceV);
void auxCrearNivel1_4(Arbol a, IndiceVector4 indiceV);
```

- Estas funciones están definidas en `indice_vector4.c`.

Para almacenar las estructuras anteriores se han definido dos tipos.

- El tipo `IndiceVector2` está definido en el fichero `tipos2.h` y contiene los siguientes campos:

- Los siguientes campos son vectores de datos que almacenan los niveles de `IndiceAVL2`. Los vectores cuyo tipo es `ListaIndicePlano` replican los nodos negros de `IndiceAVL2`. El vector cuyo tipo es `ListaID` replica los nodos rojos.

```
ListaID vector_ids;
ListaIndicePlano indiceNivel2;
ListaIndicePlano indiceNivel1;
```

- Para finalizar estos últimos campos se corresponden con el **tamaño** de los vectores anteriores.

```
int numeroEstrellasIndice;
int numeroNodosNivel2;
int numeroNodosNivel1;
```

- El tipo `IndiceVector4` está definido en el fichero `tipos4.h` y contiene los siguientes campos:

- Los siguientes campos son vectores de datos que almacenan los niveles de `IndiceAVL4`. Los vectores cuyo tipo es `ListaIndicePlano` replican los nodos negros de `IndiceAVL4`. El vector cuyo tipo es `ListaID` replica los nodos rojos.

```
ListaID vector_ids;
ListaIndicePlano indiceNivel4;
ListaIndicePlano indiceNivel3;
ListaIndicePlano indiceNivel2;
ListaIndicePlano indiceNivel1;
```

- Para finalizar estos últimos campos se corresponden con el **tamaño** de los vectores anteriores.

```
int numeroEstrellasIndice;
int numeroNodosNivel4;
int numeroNodosNivel3;
int numeroNodosNivel2;
int numeroNodosNivel1;
```

También se han implementado funciones para liberar la memoria reservada para **índice vector**.

- La función que libera la memoria de `IndiceVector2` está definida en `indice_vector2.c` y su cabecera es:

```
void PodarIndiceVector_2(IndiceVector2 *indiceV);
```

- La función que libera la memoria de `IndiceVector4` está definida en `indice_vector4.c` y su cabecera es:

```
void PodarIndiceVector_4(IndiceVector4 *indiceV);
```

Ambas estructuras, **índice AVL** e **índice vector**, son estructuras **equivalentes**. La estructura que se utilizará en el procesado de estrellas madre será el índice vector, por tanto, la estructura índice AVL puede ser eliminada.

En la tabla 8.1 se muestra el tiempo necesario para hacer la transformación de índice AVL a índice vector.

Ficheros Crowding	Dos condiciones	Cuatro condiciones
crow_m1_nuevo2.databs	0,0223	0,0556
f10.comp.calabs	1,1206	2,7178

Tabla 8.1: Tabla de tiempos. Transformación de la estructura Índice AVL a Índice Vector.

A continuación se muestra la transformación de la estructura de la figura 5.2 a índice vector. Recordar que para construir la estructura índice vector **hay que recorrer la estructura índice AVL de la misma manera en la que recorremos un árbol AVL en inorden** teniendo en cuenta que cada **nodo negro tiene tres enlaces** (campos izquierdo, siguiente_nivel y derecho). Para recorrer un **nodo negro** se empieza por izquierdo, se continúa con siguiente_nivel y se finaliza con derecho.

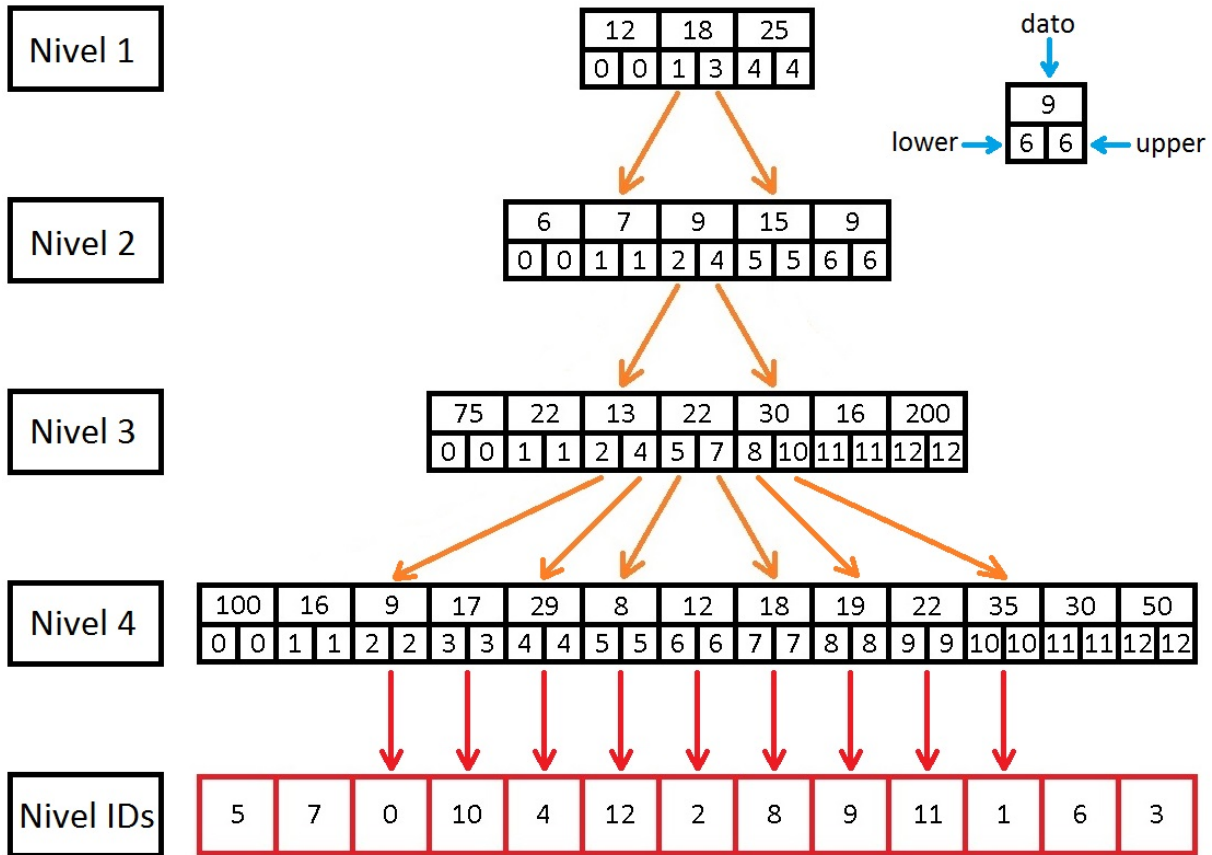


Figura 8.1: Transformación de la estructura mostrada en la figura 5.2 a Índice Vector.

Dado un dato de un determinado nivel (Nivel 1 - Nivel 3), los **enlaces naranjas** señalan los datos que este tiene asociado en el siguiente nivel. En el nivel 4, por tratarse del último dato, los **enlaces rojos** señalan a identificadores de estrellas crowding. Se han omitido muchos enlaces naranjas y rojos por simplicidad. En realidad, la estructura no tiene estos enlaces. El papel de estos enlaces lo hacen los campos `lower` y `upper` de `ListaIndicePlano`.

8.9 Procesado de estrellas madre: Dispersión

Con la estructura índice vector podemos comenzar con el procesado de estrellas madres.

Las funciones que realizan el procesado de estrellas madre están definidas en `dispersion2.c` y `dispersion4.c`. Sus ficheros de cabecera asociados son `dispersion2.h` y `dispersion4.h`.

El contenido de estos ficheros varía dependiendo de la versión (`OPENMP` o `CUDA-OPENMP`).

- En la versión `CUDA-OPENMP` tendremos las siguientes funciones de procesado:

```
__global__ void crowd_avl_2index_2cond_cuda_noUM_0(...);
void crowd_avl_2index_2cond_openmp_0(...);
__global__ void crowd_avl_4index_4cond_cuda_noUM_0(...);
void crowd_avl_4index_4cond_openmp_0(...);
```

- En la versión **OPENMP** sólo tendremos las funciones cuyo nombre incluye openmp.

```
void crowd_avl_2index_2cond_openmp_0(...);
void crowd_avl_4index_4cond_openmp_0(...);
```

Las **funciones de procesamiento OpenMP** necesitan los siguientes parámetros:

- **ListaDatosMadre**
- **ListaDatosMadreExp**
- **Estructura índice vector**: usaremos **IndiceVector2** si la función de procesamiento que vamos a ejecutar es de 2 condiciones (`avl_2index_2cond`). Por otra parte usaremos **IndiceVector4** si la función de procesamiento que vamos a ejecutar es de 4 condiciones (`avl_4index_4cond`).
- **ListaDatosCrowdingParcial**
- **ParametrosEjecucion**: contiene los márgenes de error para cada característica, el número mínimo de estrellas a encontrar, la tolerancia y el número de índices.
- **ListaDatosDispersoReal**
- **ListaDatosDispersoEntero**
- **ListaUsoEstrellasCrowding**

Las **funciones de procesamiento CUDA** necesitan dos parámetros adicionales:

- **tam_total**: debido a restricciones de memoria, el procesamiento de estrellas madre mediante CUDA, se realiza por lotes. Con este parámetro especificamos el tamaño del lote a procesar.
- **states**: este parámetro se usa para seleccionar estrellas crowding de forma aleatoria durante el proceso de dispersión.

Las cuatro funciones anteriores hacen uso de las funciones de búsqueda propias de la estructura índice vector. Las funciones de búsqueda están definidas en los ficheros `indice_vector2.c` e `indice_vector4.c` y sus cabeceras son respectivamente:

```
void BuscarEstrellasIndiceVector_2
(
    IndiceVector2 indiceV,
    tipoDato *rangos,
    int *contador,
    ListaID list,
    int estrellaDispersoEntero,
    ListaDatosDispersoEntero *listaDatosDispersoEntero
);

void BuscarEstrellasIndiceVector_4
(
    IndiceVector4 indiceV,
    tipoDato *rangos,
    int *contador,
    ListaID list,
    int estrellaDispersoEntero,
    ListaDatosDispersoEntero *listaDatosDispersoEntero
);
```

Las funciones de búsqueda anteriores sustituyen a los bucles for (líneas 3-11 y líneas 15-23) del pseudocódigo del apartado 2.2.

Los parámetros que necesitan las funciones de búsqueda son:

- Estructura **índice vector**:
 - Usaremos `IndiceVector2` con la función de búsqueda `BuscarEstrellasIndiceVector_2`.
 - Usaremos `IndiceVector4` con la función de búsqueda `BuscarEstrellasIndiceVector_4`.
- **rangos**: vector que contiene los rangos a buscar en cada índice. Cada característica tiene su valor inferior ($valor_madre_c - \varepsilon_c$) y su valor superior ($valor_madre_c + \varepsilon_c$).
- **contador**: número de estrellas crowding que han sido seleccionadas en la búsqueda con los valores del parámetro **rangos**.
- **list**: lista con los identificadores de las estrellas crowding que han resultado seleccionadas en la búsqueda.
- **estrellaDispersoEntero**: identificador de la estrella madre que está siendo dispersada.
- **listaDatosDispersoEntero**: este parámetro guarda los indicadores que proporcionan información del resultado del proceso de dispersión de las estrellas madre. En concreto el tercer indicador puede verse modificado mientras se realizan las búsquedas.

Las funciones de búsqueda anteriores hacen uso de la búsqueda binaria. Para explicar brevemente estas funciones tomemos como referencia la figura 8.1. La función de búsqueda comenzaría aplicando la búsqueda binaria sobre **todo** el vector del nivel 1 obteniendo de esta forma un índice inferior y un índice superior. El siguiente paso sería recorrer el subvector de nivel 1 delimitado por ambos índices, haciendo búsquedas binarias en los vectores de los niveles inferiores, **usando como extremos en las búsquedas los campos lower y upper**. Después de haber realizado la cuarta búsqueda binaria **usando los campos lower y upper** copiamos los identificadores de las estrellas crowding en la lista de estrellas seleccionadas.

Las funciones que realizan la búsqueda binaria de los índices inferior y superior están definidas en `busqueda_binaria.c`. El fichero de cabecera asociado es `busqueda_binaria.h`. Las cabeceras de las funciones son:

```
int busquedaBinariaIndiceVectorInferior
(
    ListaIndicePlano lista,
    int lower,
    int upper,
    tipoDato dato
);

int busquedaBinariaIndiceVectorSuperior
(
    ListaIndicePlano lista,
    int lower,
    int upper,
    tipoDato dato
);
```

Los parámetros que reciben estas funciones son:

- **lista**: vector de datos sobre el que se efectúa la búsqueda.
- **lower**: índice inferior del vector sobre el que se comienza la búsqueda.
- **upper**: índice superior del vector sobre el que se comienza la búsqueda.
- **dato**: dato que hay que buscar dentro del vector entre los índices lower y upper.

Estas funciones devuelven un índice del vector o error. En principio las funciones devuelven el índice del vector que ocupa el dato exacto buscado. Puede darse el caso de que el dato exacto que buscamos no esté en el vector, en ese caso se devuelve el índice del dato compatible. Si no existe dato compatible con el dato exacto se devuelve error.

Las funciones que realizan la búsqueda binaria controlan los signos de las condiciones de comparación (líneas 5-8 y líneas 17-20 del pseudocódigo del apartado 2.2).

- La función que busca el índice inferior (`busquedaBinariaIndiceVectorInferior`) controla si el signo es un **mayor (rango abierto)** o un **mayor igual (rango cerrado)**.
- La función que busca el índice superior (`busquedaBinariaIndiceVectorSuperior`) controla si el signo es un **menor (rango abierto)** o un **menor igual (rango cerrado)**.

El código de ambas funciones incluye el comentario correspondiente indicando esta situación.

8.10 Escritura de resultados en disco.

Con el proceso de dispersión acabado lo único que queda por hacer es escribir los resultados en disco. El programa escribe en disco tres ficheros:

- **Fichero dispersado:** los datos que debe contener este fichero se encuentran almacenados en

```
ListaDatosDispersoReal
ListaDatosDispersoEntero
```

- **Fichero crowding de salida:** el fichero crowding de salida contiene la misma información que el fichero crowding de entrada y un indicador que muestra el número de veces que se usó cada estrella en el proceso de dispersión. El orden de las estrellas crowding en el fichero es el que resultado de recorrer la estructura índice vector. Para crear este fichero necesitamos:

```
IndiceVector2 o IndiceVector4 // Estructura índice vector
ListaUsoEstrellasCrowding
ListaDatosCrowding
```

- **Fichero de selecciones:** este fichero es un extracto del fichero dispersado y su contenido fue descrito en el apartado 3.2.

Las variables que se muestran a continuación almacenan los datos de salida. Las reservas se realizan antes de llamar a las funciones de procesado. Se han definido una serie de constantes para ello en `constantes.h`.

```
ListaDatosDispersoReal
ListaDatosDispersoEntero
ListaUsoEstrellasCrowding
```

- La constante utilizada para reservar la memoria de los valores reales:

```
NUM_COL_DISPERSO_FLOAT
```

- La constante utilizada para reservar la memoria de los valores enteros:

```
NUM_COL_DISPERSO_INT
```

- Para cada estrella crowding de salida solo es necesario un valor entero, es por ello, que no se ha definido constante.

Para poder acceder a los distintos valores de las estrellas dispersadas, se han definido una serie de constantes en el fichero `constantes.h`.

- Para los valores reales:

```
COL_DISPERSO_M1
COL_DISPERSO_M2
COL_DISPERSO_EDAD
COL_DISPERSO_METALICIDAD
COL_DISPERSO_X
COL_DISPERSO_Y
```

- Para los valores enteros:

```
COL_DISPERSO_VUELTA
COL_DISPERSO_N
COL_DISPERSO_OVER_BUFFER
```

Para acceder al primer valor (M1) de una **estrella dispersada** en `listaDatosDispersoReal` haremos lo siguiente:

```
listaDatosDispersoReal[i * NUM_COL_DISPERSO_FLOAT + COL_DISPERSO_M1]
// siendo i el número de estrella dispersada
```

Las funciones para escribir el fichero crowding de salida están definidas en `salida_crowding2.c` y `salida_crowding4.c`.

```
void imprimir_fichero_crowding_ordenado_con_annotaciones_indice_vector_2
(
    ListaDatosCrowding,
    IndiceVector2,
    ListaUsoEstrellasCrowding,
    ParametrosEjecucion
);

void imprimir_fichero_crowding_ordenado_con_annotaciones_indice_vector_4
(
    ListaDatosCrowding,
    IndiceVector4,
    ListaUsoEstrellasCrowding,
    ParametrosEjecucion
);
```

Las funciones para escribir el fichero dispersado y de selecciones están definidas en `salida_dispersa.c`.

```
void imprimir_fichero_disperso
(
    ListaDatosDispersoEntero,
    ListaDatosDispersoReal,
    ParametrosEjecucion
);

void imprimir_fichero_selecciones
(
    ListaDatosDispersoEntero,
    ParametrosEjecucion
);
```

Una vez concluida la escritura de los ficheros de salida se puede liberar toda la memoria que aún está reservada.

8.11 Modo de uso: Compilación y ejecución

En este apartado veremos la forma de utilizar esta implementación, teniendo en cuenta que se cumplen los requisitos de instalación explicados en el apartado 8.2.

Todo el código fuente relativo a esta implementación se encuentra en el directorio `/cli/gpu/` del dispositivo de almacenamiento.

- El código de la versión **CUDA-OPENMP** en el directorio `/cli/gpu/CUDA-OPENMP`.
- El código de la versión **OPENMP** en el directorio `/cli/gpu/OPENMP`.

La generación de los ejecutables de las distintas versiones (CUDA-OPENMP y OPENMP) se realizará a través de un fichero Makefile. En los ficheros Makefile se pueden realizar los siguientes ajustes:

- Establecer el **número máximo de estrellas crowding** que se pueden seleccionar en las búsquedas. Recordemos que si en una búsqueda se alcanza este límite se abortara la búsqueda (informando de esta situación en la última columna del fichero dispersado) continuando con el proceso de dispersión. La variable definida en los ficheros makefile es:

- `MAX_ESTRELLAS_SELECCIONADAS`: esta variable que se ha definido en los ficheros `makefile` que acompañan a las versiones `CUDA-OPENMP` y `OPENMP` se corresponde con una constante en el código C que **limita el tamaño del vector** en el que se guardan las estrellas crowding que se van seleccionando en las búsquedas. En ningún caso debe confundirse con el parámetro `N`.
- En el caso de la versión `CUDA-OPENMP` se puede establecer la **compute capability** (SM version) de la GPU que se va a usar. La GPU Titan Black que hemos usado en las pruebas trabaja con el chip **GK110B** y su compute capability es **3.5**. La compute capability dependerá de la tarjeta que usemos. La compute capability de las distintas GPUs se puede encontrar en la página de CUDA. Las variables definidas en el fichero `makefile` son:
 - `ARQUITECTURE`
 - `CODE`

Por tanto, para generar el ejecutable de las distintas versiones (**CUDA-OPENMP** y **OPENMP**) lo único que hay que hacer es ejecutar **make** en una consola.

Para ejecutar el binario generado, ejecutamos:

```
./nombre_ejecutable fichero_parametros
```

- `nombre_ejecutable`: nombre del fichero binario generado por el comando **make**.
- `fichero_ parámetros`: nombre del fichero de parámetros o ruta hasta el mismo.

Para finalizar, en la **línea 18** del fichero de parámetros de la versión **CUDA-OPENMP** se indica el identificador de GPU que se va a utilizar. Si tenemos más de una GPU en el sistema es conveniente compilar el fichero fuente `deviceQuery.c` (incluido con los Nvidia CUDA Samples) y ejecutar el binario generado para asegurarnos de que la GPU que usamos es la correcta.

8.12 Prueba de rendimiento

En las siguientes tablas y gráficas se muestran las pruebas de rendimiento realizadas al código de esta implementación. Este hecho se indica en la fila de cabecera en cada una de las tablas mediante tipo de versión (**avl**).

La manera de presentar las pruebas es la misma que la introducida en el apartado 3.1 y ha sido explicada con mayor detenimiento en el capítulo 4.

En el siguiente apartado se utilizarán los resultados obtenidos en el capítulo 4 y los que aquí se presentan para realizar una comparación y determinar el alcance de la mejora.

Para las pruebas recogidas en la **tabla 8.2** se han utilizado como ficheros de datos el **primer par de ficheros** cuyo **tamaño** puede ser consultado en el **apartado 3.1**. El número de condiciones que se han utilizado para realizar las búsquedas y selecciones de estrellas crowding es **2**.

Estrellas procesadas	SECUENCIAL	OPEN-MP	CUDA
10	0,0005	0,0388	0,0108
100	0,0047	0,0375	0,0209
1 000	0,0436	0,046	0,0326
10 000	0,4312	0,109	0,0407
100 000	4,3027	0,755	0,4026
1 000 000	46,5981	7,2063	7,6063
1 500 000	64,0458	10,7505	10,7256

Tabla 8.2: Rendimiento de la implementación: **dos** condiciones y **primer par** de ficheros.

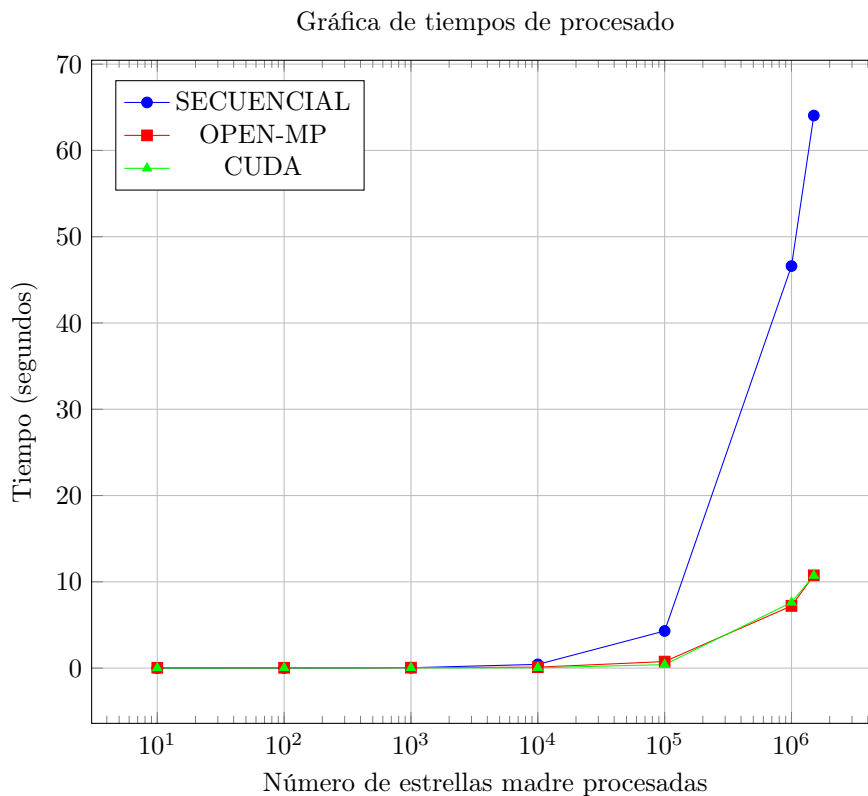


Figura 8.2: Gráfica de tiempos de procesado. Tabla 8.2.

En la figura 8.2 se puede observar que cuando se procesa un número de estrellas madre relativamente pequeño (hasta 10^4) **no existe diferencia** entre usar paralelismo (OPEN-MP y CUDA) o no usarlo (SECUENCIAL).

El punto de ruptura entre usar paralelismo o no usarlo se **manifiesta ligeramente** por primera vez al procesar 10^5 estrellas madre. Este punto de ruptura se **consolida** cuando procesamos 10^6 estrellas madre.

Con los resultados obtenidos y a la vista de la figura 8.2 parece no haber diferencias entre usar paralelismo a nivel de CPU o a nivel de GPU. Si consultamos los datos de la tabla 8.2 los tiempos resultan ser iguales.

Con los datos de los que disponemos no podemos hacer distinciones sobre cuál es el tipo de paralelismo ganador.

Para las pruebas recogidas en la **tabla 8.3** se han utilizado como ficheros de datos el **primer par de ficheros** cuyo **tamaño** puede ser consultado en el **apartado 3.1**. El número de condiciones que se han utilizado para realizar las búsquedas y selecciones de estrellas crowding es **4**.

• Número de Condiciones	4		
• Ficheros de datos	Primer par (Ver apartado 3.1)		
• Tipo de versión	avl		
Estrellas procesadas	SECUENCIAL	OPEN-MP	CUDA
10	0,0013	0,0636	0,03
100	0,0107	0,0683	0,0488
1 000	0,0991	0,0815	0,0756
10 000	0,9544	0,229	0,0798
100 000	9,6588	1,7304	0,5578
1 000 000	111,7927	20,4663	8,757
1 500 000	167,2885	30,6436	12,5456

Tabla 8.3: Rendimiento de la implementación: **cuatro** condiciones y **primer par** de ficheros.

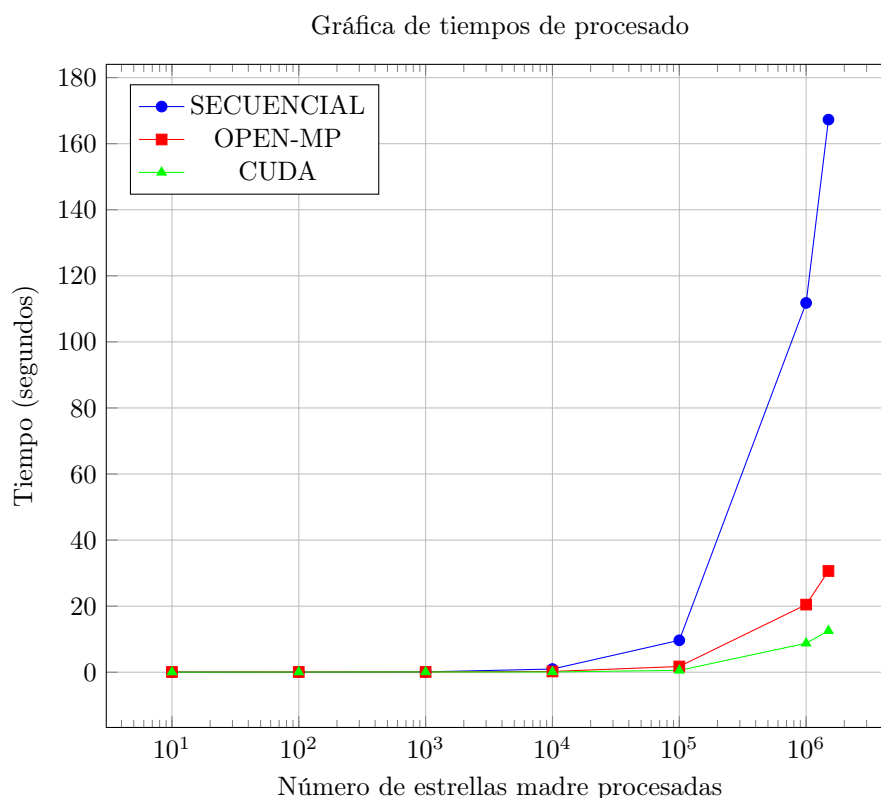


Figura 8.3: Gráfica de tiempos de procesado. Tabla 8.3.

En la figura 8.3 se puede observar que cuando se procesa un número de estrellas madre relativamente pequeño (hasta 10^4) **no existe diferencia** entre usar paralelismo (OPEN-MP y CUDA) o no usarlo (SECUENCIAL).

El punto de ruptura entre usar paralelismo o no usarlo se **manifiesta ligeramente** por primera vez al procesar 10^5 estrellas madre. Este punto de ruptura se **consolida** cuando procesamos 10^6 estrellas madre.

En este caso existe diferencia entre usar paralelismo a nivel de CPU y a nivel de GPU. Esta diferencia

se manifiesta en la figura 8.3 cuando se procesan 10^6 estrellas madre.

La figura 8.3 indica que CUDA es la ganadora en este caso.

Para las pruebas recogidas en la **tabla 8.4** se han utilizado como ficheros de datos el **segundo par de ficheros** cuyo **tamaño** puede ser consultado en el **apartado 3.1**. El número de condiciones que se han utilizado para realizar las búsquedas y selecciones de estrellas crowding es **2**.

Estrellas procesadas	SECUENCIAL	OPEN-MP	CUDA
10	0,003	3,7045	0,09
100	0,0295	3,4671	0,1481
1 000	0,2999	3,7511	0,2373
10 000	3,0211	4,2166	0,4106
100 000	30,163	8,6993	5,1249
1 000 000	304,2642	55,7874	52,8832
10 000 000	–	529,1155	528,6105
16 000 000	–	866,4046	845,7623

Tabla 8.4: Rendimiento de la implementación: **dos** condiciones y **segundo par** de ficheros.

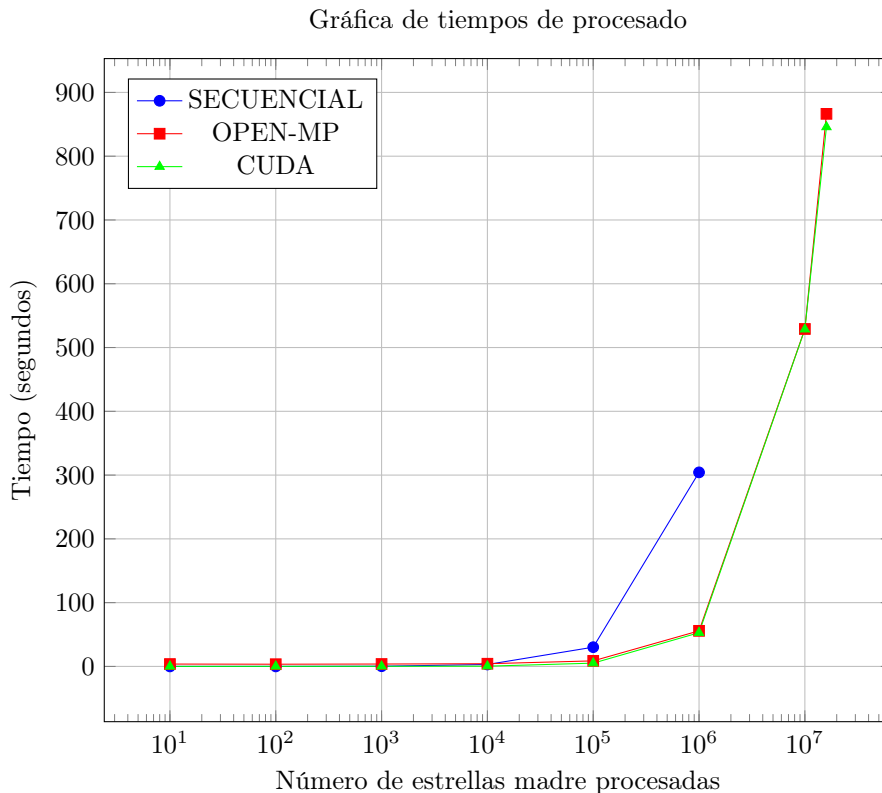


Figura 8.4: Gráfica de tiempos de procesado. Tabla 8.4.

En la figura 8.4 se puede observar la misma situación que en la figura 8.2, en cuanto a comparaciones se refiere.

Con los datos de los que disponemos no podemos hacer distinciones sobre cuál es el tipo de paralelismo ganador.

Para las pruebas recogidas en la **tabla 8.5** se han utilizado como ficheros de datos el **segundo par de ficheros** cuyo **tamaño** puede ser consultado en el **apartado 3.1**. El número de condiciones que se

han utilizado para realizar las búsquedas y selecciones de estrellas crowding es 4.

Estrellas procesadas	SECUENCIAL	OPEN-MP	CUDA
10	0,0064	4,5686	0,1332
100	0,065	4,7227	0,2156
1 000	0,6738	4,7142	0,33
10 000	6,7514	5,8857	0,464
100 000	67,6954	16,4766	4,4536
1 000 000	684,9019	125,6757	46,5718
10 000 000	–	1 490,7665	557,6948
16 000 000	–	–	923,8403

Tabla 8.5: Rendimiento de la implementación: **cuatro** condiciones y **segundo par** de ficheros.

Gráfica de tiempos de procesado

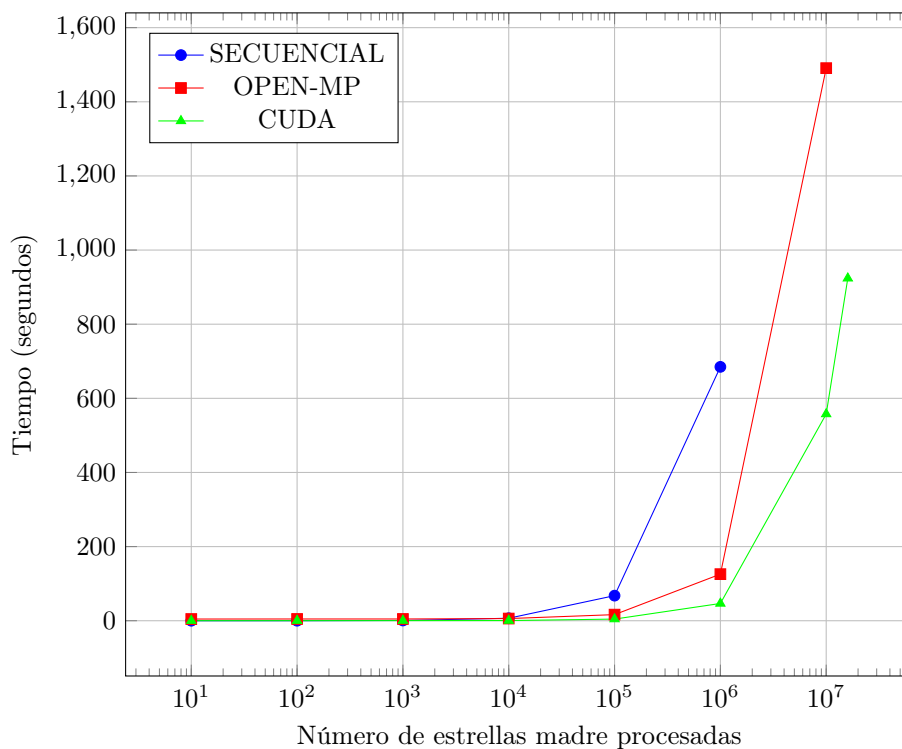


Figura 8.5: Gráfica de tiempos de procesado. Tabla 8.5.

En la figura 8.5 se puede observar la misma situación que en la figura 8.3, en cuanto a comparaciones se refiere.

La figura 8.5 indica que CUDA es la ganadora en este caso.

8.13 Conclusiones

Para finalizar se realizará una comparación de los tiempos obtenidos entre el pseudocódigo del apartado 2.2 y la implementación de este apartado.

Los tiempos obtenidos utilizando el pseudocódigo del apartado 2.2 son inasumibles. Por tanto, se introduce la estructura explicada en el capítulo 5 y en el caso de esta implementación dicha estructura sufre la transformación explicada en el apartado 8.8. La estructura usada se construye para dos y cuatro condiciones, por ello, haremos dos comparaciones entre ambas versiones:

- Para **dos condiciones** tomemos como referencia las **tablas 5.3 y 8.4, columna SECUENCIAL** (prueba secuencial) y la fila correspondiente al **procesamiento de 10000 estrellas madres**. En el caso de la tabla 5.3 el tiempo es **939.9352 segundos** y en la tabla 8.4 es **3.0211 segundos**. Por tanto la **aceleración o speedup** obtenido es de **311**.
- En el caso de **cuatro condiciones** tomemos como referencia las **tablas 5.4 y 8.5, columna SECUENCIAL** y la fila que corresponde al **procesamiento de 10000 estrellas madres**. En el caso de la tabla 5.4 el tiempo es **1175.1146 segundos** y en la tabla 8.5 es **6.7514 segundos**. En este caso el speedup obtenido es de **174**.

Una vez realizada esta comparación podemos afirmar que la implementación explicada en este apartado es bastante más rápida y eficiente que el pseudocódigo del apartado 2.2.

Recordemos que además sobre esta implementación, se ha utilizado paralelismo a nivel de CPU y a nivel de GPU. Se pueden realizar dos comparaciones más y para ello se utilizarán las **tablas 8.4 (2 condiciones) y 8.5 (4 condiciones)**.

- La primera comparación consiste en comparar la ejecución secuencial (columna SECUENCIAL) con la ejecución que usa paralelismo de CPU (columna OPENMP). Utilizaremos en ambas tablas las filas que suponen el procesamiento de **1 millón de estrellas madres**.
 - En el caso de la **tabla 8.4** tenemos, en la **columna SECUENCIAL 304.2642 segundos** y en la **columna OPEN-MP 55.7874 segundos**.
 - En el caso de la **tabla 8.5** tenemos, en la **columna SECUENCIAL 684.9019 segundos** y en la **columna OPEN-MP 125.6757 segundos**.
 - En ambos casos el speedup obtenido es el mismo, **5.45**. Los speedup obtenidos dependen del número de núcleos que tenga el procesador utilizado.
- La segunda y última comparación consiste en comparar los resultados obtenidos ambos tipos de paralelismos (CPU y GPU). Para ello nos fijamos en la columnas **OPEN-MP y CUDA**.
 - En el caso de la **tabla 8.4** para realizar la comparación nos fijamos en la fila que conlleva el procesamiento de **16 millones de estrellas madres**. En el caso de **OPEN-MP** tenemos **866.4046 segundos** y en el caso de **CUDA 845.7623 segundos**. El speedup es **1.02**, muy cercano a 1. Por tanto **sin tener más datos** podemos decir que no hay diferencias notables entre ambas versiones.
 - En el caso de la **tabla 8.5** para realizar la comparación nos fijamos en la fila que conlleva el procesamiento de **10 millones de estrellas madres**. En el caso de **OPEN-MP** tenemos **1490.7665 segundos** y en el caso de **CUDA 557.6948 segundos**. El speedup es **2,67**.
- Los speedup obtenidos anteriormente pueden variar dependiendo de los procesadores (CPU) y núcleos gráficos (GPU) utilizados.

Capítulo 9

Comparación de las implementaciones

En los capítulos 7 y 8 se han presentado dos implementaciones que resuelven el problema presentado introducido de forma general en la sección 1.1 y de forma particular en la sección 2.2. Ambas implementaciones ofrecen funcionalidades diferentes, pero, son capaces de resolver el mismo problema. En este capítulo se realiza una comparación entre ambas implementaciones en cuanto al tiempo de ejecución se refiere.

Para comparar las implementaciones de los capítulos 7 y 8 se usaran los resultados obtenidos en las secciones 7.5 y 8.12 respectivamente.

- Para la implementación del capítulo 7 se han usado las siguientes tablas:
 - La tabla 7.3 de tiempos correspondiente a las pruebas con ficheros grandes y dos condiciones. Se ha usado la segunda columna.
 - La tabla 7.4 de tiempos correspondiente a las pruebas con ficheros grandes y cuatro condiciones. Se ha usado la segunda columna.
- Por otra parte para la implementación del capítulo 8 se han usado las siguientes tablas:
 - La tabla 8.4 de tiempos correspondiente a las pruebas con ficheros grandes y dos condiciones. Se han usado las columnas OPEN-MP y CUDA.
 - La tabla 8.5 de tiempos correspondiente a las pruebas con ficheros grandes y cuatro condiciones. Se han usado las columnas OPEN-MP y CUDA.

De esta forma las tablas que se muestran a continuación tendrán cuatro columnas:

- La **primera** (Estrellas procesadas) indica el numero de estrellas madres procesadas.
- La **segunda** (OpenMP iterativo) indica el tiempo de procesado OPEN-MP de la implementación del capítulo 8.
- La **tercera** (CUDA) indica el tiempo de procesado CUDA de la implementación del capítulo 8.
- La **cuarta** (OpenMP recursivo) indica el tiempo de procesado OPEN-MP de la implementación del capítulo 7.

Para una mejor comprensión e interpretación de las tablas, cada una ellas tendrá asociada una gráfica en la que se mostrarán los tiempos obtenidos sobre el número de estrellas madre procesadas.

Para las pruebas recogidas en la tabla 9.1 se han utilizado como ficheros de datos el **segundo par de ficheros** cuyo **tamaño** puede ser consultado en el **apartado 3.1**. El número de condiciones que se han utilizado para realizar las búsquedas y selecciones de estrellas crowding es **2**.

Estrellas procesadas	OpenMP iterativo	CUDA	OpenMP recursivo
10	3,7045	0,09	0,0315
100	3,4671	0,1481	0,0545
1 000	3,7511	0,2373	0,4152
10 000	4,2166	0,4106	4,1817
100 000	8,6993	5,1249	38,2625
1 000 000	55,7874	52,8832	383,737
10 000 000	529,1155	528,6105	–
16 000 000	866,4046	845,7623	–

Tabla 9.1: Comparación entre las implementaciones del cap. 7 y cap. 8: **dos** condiciones y **segundo par** de ficheros.

En base a los resultados mostrados en la tabla 9.1 podemos hacer dos comparaciones:

- Comparación entre OpenMP iterativo y OpenMP recursivo. Tomando como referencia el procesado de un millón de estrellas madre tenemos que:
 - Para OpenMP iterativo el tiempo consumido es de **55.7874 segundos**.
 - Para OpenMP recursivo el tiempo consumido es de **383.737 segundos**.
 - Estos datos anteriores indican que OpenMP iterativo es 6.88 veces más rápido que OpenMP recursivo.

- Comparación entre OpenMP recursivo y CUDA. Tomando como referencia el procesado de un millón de estrellas madre tenemos que:
 - Para OpenMP recursivo el tiempo consumido es de **383.737 segundos**.
 - Para CUDA el tiempo consumido es de **52.8832 segundos**.
 - Estos datos anteriores indican que CUDA es 7.26 veces más rápido que OpenMP recursivo.

Gráfica de tiempos de procesado

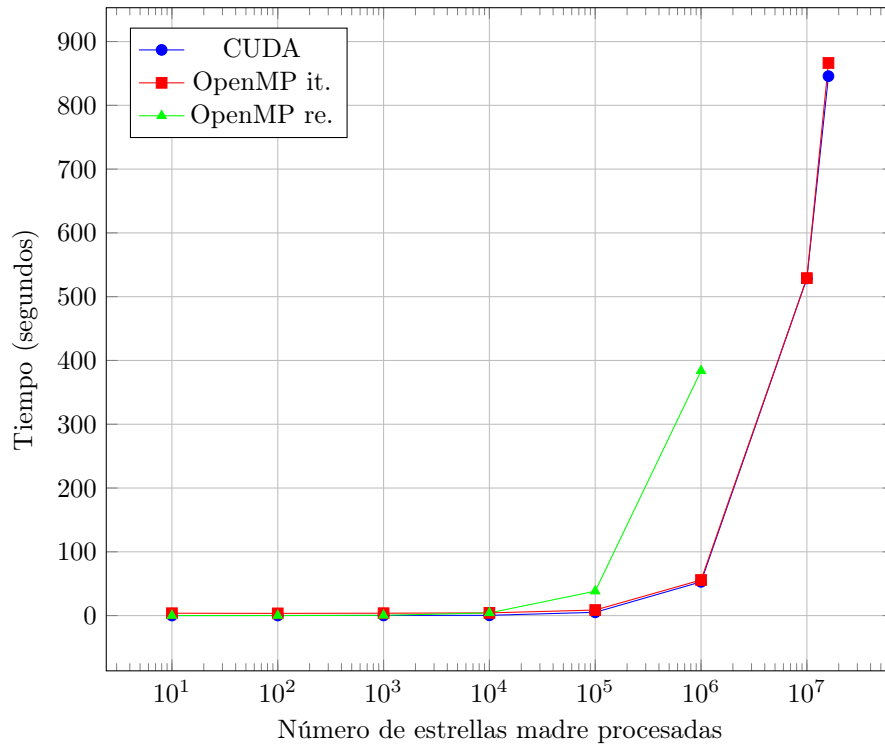


Figura 9.1: Gráfica de tiempos de procesado. Tabla 9.1.

En la figura 9.1 se puede observar que cuando se procesa un número de estrellas madre relativamente pequeño (hasta 10^4) **no existe diferencia** entre que versión usar.

El punto de ruptura entre usar la implementación OpenMP recursiva o no usarla se **manifiesta ligeramente** por primera vez al procesar 10^5 estrellas madre. Este punto de ruptura se **consolida** cuando procesamos 10^6 estrellas madre.

La figura 9.1 no aclara quien es la ganadora, si lo es OpenMP it. o CUDA, ambas parecen estar empatadas.

Para las pruebas recogidas en la tabla 9.2 se han utilizado como ficheros de datos el **segundo par de ficheros** cuyo **tamaño** puede ser consultado en el **apartado 3.1**. El número de condiciones que se han utilizado para realizar las búsquedas y selecciones de estrellas crowding es **4**.

Estrellas procesadas	OpenMP iterativo	CUDA	OpenMP recursivo
10	4,5686	0,1332	8,5115
100	4,7227	0,2156	0,1615
1 000	4,7142	0,33	1,3283
10 000	5,8857	0,464	11,1537
100 000	16,4766	4,4536	96,0288
1 000 000	125,6757	46,5718	1 024,29
10 000 000	1 490,7665	557,6948	–
16 000 000	–	923,8403	–

Tabla 9.2: Comparación entre las implementaciones del cap. 7 y cap. 8: **cuatro** condiciones y **segundo par** de ficheros.

En base a los resultados mostrados en la tabla 9.2 podemos hacer dos comparaciones:

- Comparación entre OpenMP iterativo y OpenMP recursivo. Tomando como referencia el procesado de un millón de estrellas madre tenemos que:
 - Para OpenMP iterativo el tiempo consumido es de **125.6757 segundos**.
 - Para OpenMP recursivo el tiempo consumido es de **1024.29 segundos**.
 - Estos datos anteriores indican que OpenMP iterativo es 8.15 veces más rápido que OpenMP recursivo.
- Comparación entre OpenMP recursivo y CUDA. Tomando como referencia el procesado de un millón de estrellas madre tenemos que:
 - Para OpenMP recursivo el tiempo consumido es de **1024.29 segundos**.
 - Para CUDA el tiempo consumido es de **46.5718 segundos**.
 - Estos datos anteriores indican que CUDA es 22 veces más rápido que OpenMP recursivo.

Gráfica de tiempos de procesado

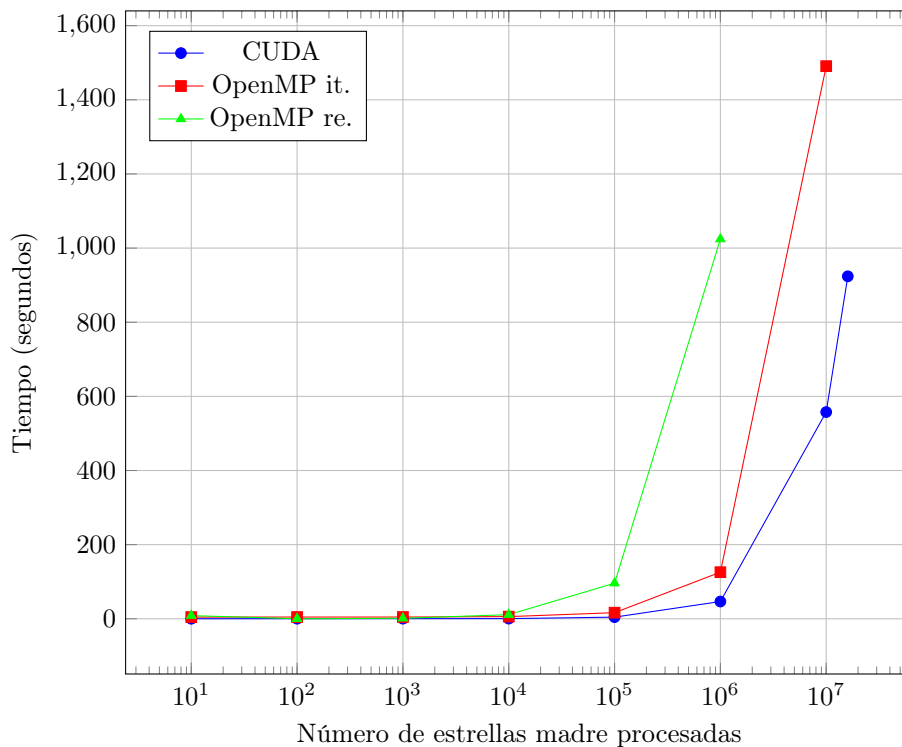


Figura 9.2: Gráfica de tiempos de procesado. Tabla 9.2.

En la figura 9.2 ocurre la misma situación que en la figura 9.1.

La figura 9.2 indica que CUDA es la ganadora.

La comparación que se ha establecido entre ambas implementaciones indica que la implementación ganadora es la explicada en el capítulo 8, pero, es importante tener presente que ambas implementaciones proporcionan funcionalidades diferentes. La implementación del capítulo 7 permite realizar las búsquedas y selección de estrellas crowding por un número arbitrario de condiciones, mientras que la implementación descrita en el capítulo 8 sólo permite 2 o 4 condiciones.

Parte II

Interfaz gráfica

Capítulo 10

Aplicación de escritorio con entorno gráfico

En este capítulo se abordará el otro de los objetivos de este Proyecto de Fin de Carrera (PFC): desarrollar un interfaz gráfica, también conocida como GUI por sus siglas en inglés. El objetivo principal del desarrollo de esta interfaz es facilitar a los usuarios la manipulación de datos de estrellas contenidos en ficheros.

Además, en esta capítulo se describe con mayor detalle cuál es el problema que se pretende solucionar, las herramientas utilizadas, su instalación, su configuración, detalles técnicos de la implementación y cómo utilizar la aplicación desarrollada.

10.1 Marco teórico

En el ámbito de la astronomía existen multitud de conceptos y términos, que conviene conocer para comprender los objetivos de la interfaz gráfica. Los conceptos y términos generales relacionados con el desarrollo de la interfaz gráfica son los siguientes: *magnitud*, *color*, *edad*, *metalicidad* de las estrellas, *diagrama color-magnitud*, *bundle*, entre otros. Algunos de estos ya fueron introducidos en la sección 1.2, otros se describirán a continuación.

El *diagrama color-magnitud* de un sistema estelar, por ejemplo, una galaxia, es una representación de las propiedades de las estrellas que lo forman, usando dos parámetros observables, uno ligado a la temperatura de la estrella (en el eje X o de abscisas) y otro relacionado con la luminosidad (en el eje Y o de ordenadas). Estos parámetros observables son el color y la magnitud, respectivamente. De hecho, el color es la diferencia de dos magnitudes. En el diagrama color-magnitud se suele representar cada estrella como un punto. Otra forma de representar el diagrama color-magnitud es usando una escala de grises o de colores indicando el número relativo de estrellas en cada punto del plano color-magnitud. En este caso se puede denominar también *diagrama de Hess*, y es útil cuando la densidad de estrellas en algunos puntos del diagrama es muy alta.

En la aplicación que hemos desarrollado, se definen zonas en el *diagrama color-magnitud* que denominaremos *bundles*. Un *bundle* es un polígono definido en el *diagrama color-magnitud*. A su vez, dentro de cada *bundle* se establece una rejilla bidimensional.

El objetivo de dividir el diagrama color-magnitud en 'bundles' y sucesivamente en zonas más pequeñas mediante la rejilla, es poder comparar la distribución de estrellas en diagramas-color magnitud observados, con la distribución en un diagrama color-magnitud sintético, obtenido a partir de modelos de evolución estelar.

Para una información más detallada sobre la obtención de diagramas color-magnitud sintéticos, se puede consultar el documento Aparicio y Gallart, «IAC-STAR: A Code for Synthetic Color-Magnitude Diagram Computation». Además, existe una aplicación disponible a través de la dirección web <http://iac-star.iac.es/cmd/index.htm>.

10.2 El problema

Por lo general, las personas tienen dificultades para procesar la información que no está presentada adecuadamente. Cambiar la representación de la información provocará que las personas la interioricen con mayor rapidez y eficiencia. Sirva como ejemplo un conjunto de parejas de números. Es más difícil interpretarlos sin una gráfica.

El principal problema que pretende resolver esta aplicación es representar los datos contenidos en ficheros y permitir su fácil manipulación.

En breve, se abordará la situación a la que el equipo colaborador del IAC debía enfrentarse antes del desarrollo de la interfaz gráfica, los objetivos concretos que debe cumplir, prototipado de la interfaz y el análisis previo al desarrollo.

10.2.1 Descripción

Antes del desarrollo de la interfaz la situación era la descrita a continuación:

El personal del IAC definía en un fichero una serie de parámetros que servían como entrada a una serie de scripts. Entre los parámetros definidos se encontraban los archivos que contenían el diagrama sintético o modelo y el diagrama observado así como la posición de los bundles en el diagrama color-magnitud, y un conjunto de intervalos de edad y metalicidad en los que se divide el diagrama color-magnitud sintético (para el cual, a diferencia del diagrama observado, conocemos las edades y metalicidades de cada estrella). Los scripts se encargaban de leer los datos y realizar los cálculos definidos en el fichero de parámetros para producir un resultado.

Los ficheros de datos que contienen los diagrama color-magnitud con la información de las estrellas tienen dos formatos. Para más información sobre estos dos formatos consultar la sección 10.4.4.

Los scripts empleados están escritos en lenguaje Python. A pesar de que estos scripts automatizan algunas tareas, otras tareas de manipulación de datos eran muy laboriosas y rudimentarias.

Entre las desventajas que presentaba el empleo de fichero de parámetros y ejecución de script se encuentran las siguientes:

- Cambios continuos en el fichero de parámetros. El sistema utilizado no era muy eficiente sobre todo cuando las labores de manipulación para obtener los resultados deseados requerían realizar ajustes continuos en el fichero de parámetros. Un ejemplo de esto era la definición de los bundle.

Definir los puntos que conforman el polígono o bundle en el diagrama color-magnitud, era una labor comparable a la de un artesano. Los puntos eran definidos de forma manual en el fichero de parámetros y su localización era verificada usando algún programa estándar de representación de gráficas.

- Escasa flexibilidad ante cambios de formato y disposición de los datos: adaptar la representación del diagrama color-magnitud implicaba modificar los scripts cuando el formato de los ficheros de datos cambiaba. Incluso si el formato no cambiaba, pero sí la distribución de los datos, era necesario modificar el script. En definitiva, era necesario tener tantos scripts como variedad de formatos y disposición de los datos de las estrellas hubiera.

Una vez conocida la situación previa al desarrollo de la interfaz, es el momento de conocer cuales son los objetivos principales que se pretende solventar.

Con la finalidad de agilizar el proceso de adquisición de requisitos, el personal del IAC proporcionó a los desarrolladores de este proyecto, un script que ejemplificaba las labores que un usuario podía realizar con el uso de la interfaz.

Por tanto *los objetivos del desarrollo de la interfaz gráfica son:*

- Leer ficheros de datos con los formatos descritos en la sección 10.4.4.2.
- Permitir definir mediante expresiones matemáticas el valor de los datos de interés.
- Representar los diagramas color-magnitud de las estrellas observadas y de las estrellas del modelo. Para ello se nos proporcionaron algunos scripts de representación de datos realizados por *Edouard Bernard*, que deben ser integrados en la interfaz gráfica.

- Permitir la definición de los puntos de los bundles, mediante eventos de ratón sobre los diagramas color-magnitud.
- Mostrar los histogramas de los bundles cuya edición ha sido finalizada. La edición de un bundle esta finalizada cuando se han definido todos sus puntos y se ha establecido su rejilla.
- Salvar a fichero la sesión actual.
- Recuperar sesiones salvadas.
- Definir intervalos de y , y contar el número de estrellas que hay en el diagrama modelo en estos intervalos.
- Generar un fichero de salida con los parámetros definidos en el script de ejemplo.

10.2.2 Prototipado de la aplicación

Las buenas prácticas en el desarrollo de aplicaciones con interfaz gráfica sugieren que se realice un boceto sobre lo que será la aplicación, con la finalidad de obtener sobre nuevos requisitos que quizás hayan sido olvidados, definir mejor los requisitos ya especificados y establecer la ubicación de los elementos. Este boceto se conoce comúnmente con el nombre de prototipo.

Gracias al empleo de esta técnica se han podido realizar algunas correcciones antes del inicio del desarrollo de la aplicación, con el consiguiente ahorro de tiempo y esfuerzo.

A continuación, se muestran los prototipos corregidos y aceptados por el personal del IAC usados para desarrollar la aplicación con interfaz gráfica.

El prototipo de la aplicación consta de tres pestañas. Cada pestaña realiza tareas diferentes e independientes.

En la figura 10.1 se ilustra la pantalla inicial de la aplicación. Por defecto, la pantalla inicial se corresponde con la pestaña más a la izquierda. Esta pestaña está destinada a la carga de ficheros y definición de los datos de interés.

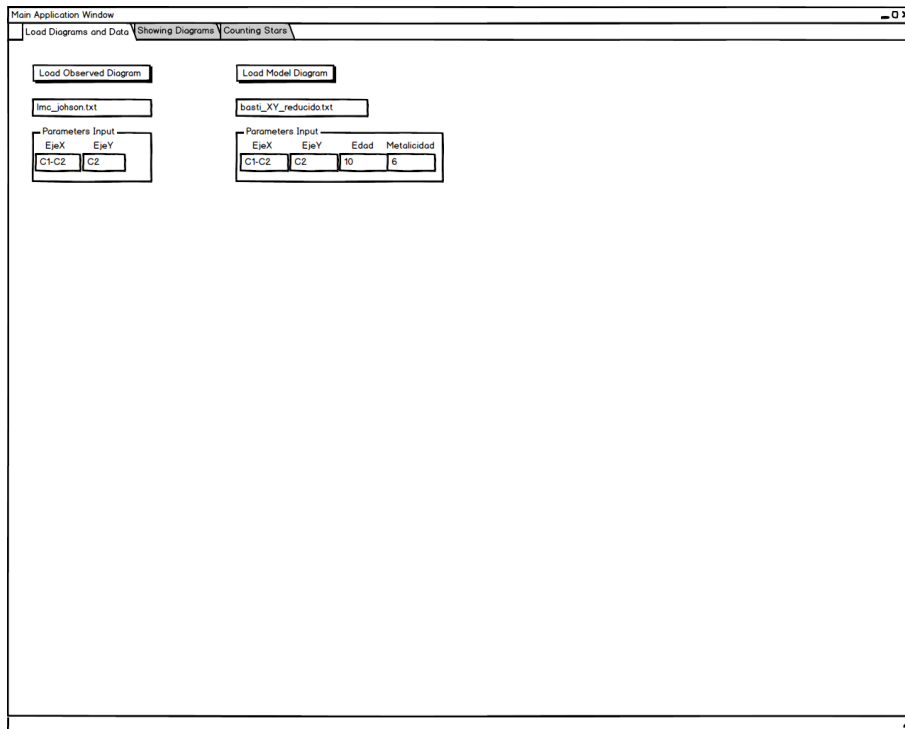


Figura 10.1: Prototipo de la pantalla inicial de la aplicación rellena de datos de ejemplo

En la figura 10.2 se puede observar la pestaña central, destinada a la representación de los diagramas color-magnitud y definición de los bundles.

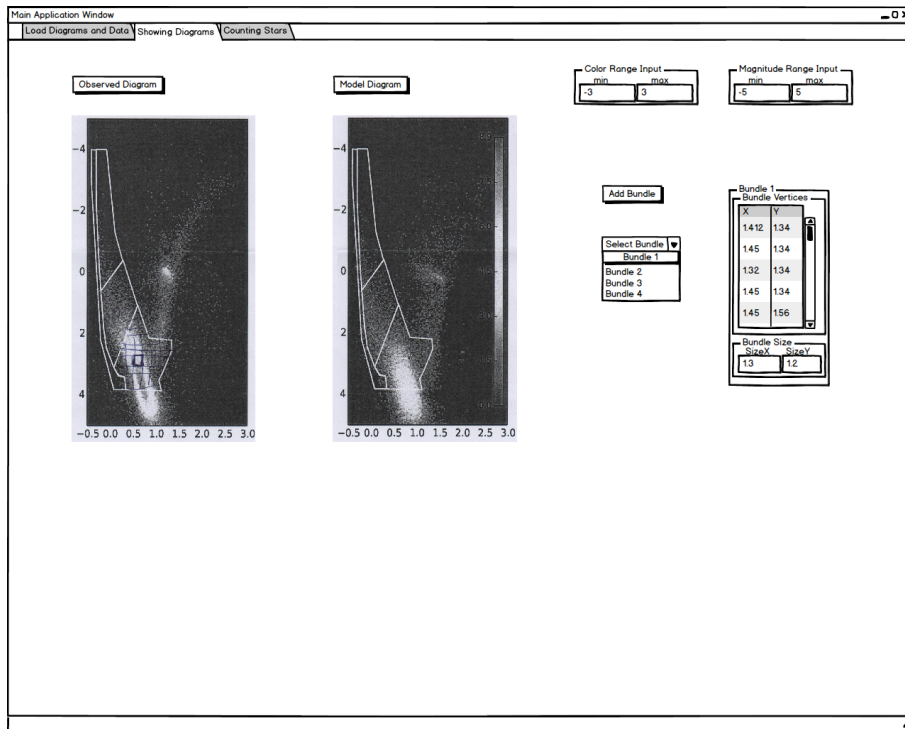


Figura 10.2: Prototipo de la pestaña central de la aplicación rellena de datos de ejemplo

En la figura 10.3 se muestra la pestaña más a la derecha, cuyo objetivo es definir el resto de parámetros necesarios.

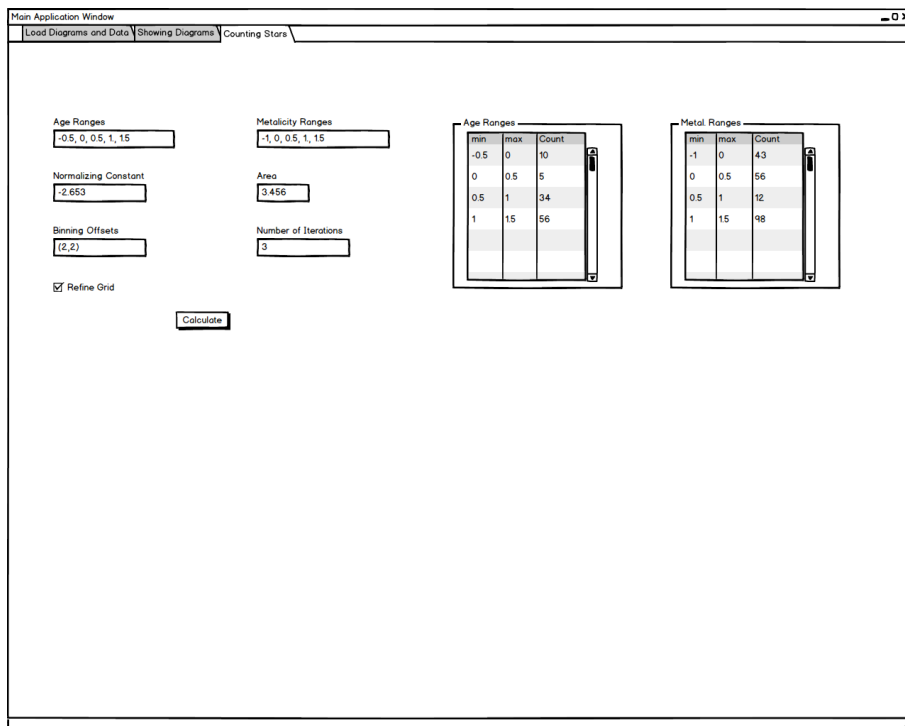


Figura 10.3: Prototipo de la última pestaña de la aplicación rellena de datos de ejemplo

10.3 Herramientas utilizadas

En este apartado se describirán las herramientas necesarias, su instalación y su configuración para poder desarrollar la interfaz y ejecutarla como usuario final.

10.3.1 Selección

El paso inicial es elegir el lenguaje base determinará las herramientas que se utilicen tanto para el desarrollo como para la ejecución de usuarios finales. Esta elección estuvo condicionada desde el inicio, porque existían scripts en Python desarrollados por *Edouard Bernard*, que debían ser integrados en la interfaz. Por tanto, el lenguaje base utilizado es Python y su versión es la *2.7.11*.

El siguiente paso es elegir alguna de las múltiples maneras de desarrollar interfaces gráficas en Python: PyQt, PyGtk, WxPython, Jython, entre otras. Se decidió elegir PyQt porque ofrece beneficios similares a los que Qt para C++. PyQt es un binding de la librería gráfica Qt para el lenguaje Python. La versión de PyQt que se ha usado es la *4.10.4*, correspondiéndose con la versión de Qt *4.8.7*.

Para la representación gráfica de los datos se ha elegido utilizar el módulo `matplotlib`.

Para manejar los ficheros de datos de formato *FITS* (*Flexible Image Transport System*), ampliamente utilizado en el mundo de la astronomía, se eligió el módulo `pyfits`.

Existen otros módulos Python elegidos en el desarrollo de esta aplicación, pero no se ha considerado relevante justificar su uso en esta memoria, aunque sí se enumeran en la sección Requisitos.

10.3.2 Requisitos

Aunque ya se han nombrado alguno de los requisitos en la sección anterior, en esta sección se enumeran los requisitos básicos de la aplicación, junto a las versiones utilizadas para que se pueda replicar con exactitud el entorno en el que la aplicación fue desarrollada.

Los requisitos básicos que tanto usuarios finales como desarrolladores deben satisfacer son los siguientes:

1. Tener instalado un interprete Python en su versión *2.7.11* para nuestro sistema operativo.
2. Disponer de los siguientes módulos Python:
 - SIP. La versión usada es la *4.16.9*.
 - PyQt4. La versión usada es la *4.10.4*.
 - ipython. La versión usada es la *3.2.0*.
 - matplotlib. La versión usada es la *1.4.3*.
 - scipy. La versión usada es la *0.17.1*.
 - pyfits. La versión usada es la *3.3*.
 - numpy. La versión usada es la *1.9.2*.
 - numexpr. La versión usada es la *2.4.3*.
 - packaging. La versión usada es la *15.3*.
 - sympy. La versión usada es la *0.7.6.1*.
3. Tener accesible el comando `pyrcc4` en la variable PATH del sistema.

10.3.3 Sugerencias

A continuación, se sugiere el uso de software adicional que permita facilitar el desarrollo futuro de esta aplicación, satisfacer los requisitos y generar distribuciones de la aplicación para sistemas concretos.

Las sugerencias de uso son las siguientes:

- Para los usuarios y desarrolladores se aconseja utilizar *Anaconda*.
- Para los desarrolladores que desee añadir nuevas funcionalidades se aconseja utilizar *PyCharm*.

- Para generar un distribución de la aplicación se aconseja utilizar *PyInstaller*.

El software sugerido es el siguiente:

- *Anaconda*: es una distribución Python que proporciona un instalador gratuito fácil de usar, que contiene una colección de más de 720 módulos de código abierto.

El uso de *Anaconda* no es obligatorio, sin embargo, es muy recomendable puesto que es muy fácil de usar y evitará realizar otras configuraciones de forma manual.

La mayoría de los paquetes requeridos son instalados por defecto. Sin embargo, los módulos `pyfits`, `pyqt` (alias del módulo `PyQt4`) y `sip` (alias del módulo `SIP`) no están incluidos por defecto.

La versión del instalador *Anaconda* que se ha usado es la *2.3.0*.

- *PyCharm*: es un Integrated Development Environment (IDE) enfocado a Python.
- *PyInstaller*: es un módulo Python que permite crear un ejecutable del script Python pasado como parámetro.

El ejecutable generado es dependiente del sistema operativo y de la arquitectura usada.

10.3.4 Instalación

En esta sección se indica como realizar la instalación de los requisitos básico y el software sugerido. Se recomienda utilizar las versiones mencionadas.

La instalación básica, sin usar el software sugerido, comprende los siguientes puntos:

1. Python: se puede obtener en su página oficial (<https://www.python.org/downloads/>), eligiendo el sistema operativo y versión *2.7.11*. El proceso de instalación es el habitual para el sistema operativo elegido.
2. Módulos: para instalar los módulos especificados el usuario o desarrollador pueden consultar la documentación oficial de Python, disponibles en la siguiente dirección web: <https://docs.python.org/2.7/installing/>. Para facilitar esta labor se ha ofrecido el fichero `requirements.txt` localizado en el directorio `/gui/configuration/`.

La instalación del entorno usando *Anaconda*, comprende los siguientes pasos:

1. *Anaconda*: la información de cómo descargar e instalar *Anaconda* se puede obtener, consultando su documentación oficial, disponible en la siguiente dirección web: <https://docs.continuum.io/anaconda/install>. En el proceso de instalación es importante elegir la modificación o personalización de la variable `PATH` de nuestro sistema operativo.
2. El resto de pasos se abordarán en mayor detalle en este documento en la sección Configuración.

Si se ha decidido desarrollar la aplicación usando *PyCharm*, es necesario seguir los siguientes pasos:

1. Dirigirse a la dirección web: <https://www.jetbrains.com/pycharm/download/>.
2. Descargar el instalador para el sistema operativo elegido.
3. Seguir las instrucciones de nuestro sistema operativo mostradas en la propia página web, en el enlace *Installation Instructions*.

Si se ha decidido generar una distribución usando *PyInstaller*, es necesario instalar el módulo Python *PyInstaller*, tal y como indica la dirección web del paso 2 de la instalación básica.

10.3.5 Configuración

En esta sección se explican las configuraciones necesarias que el usuario o desarrollador deben aplicar en su sistema para lograr ejecutar esta aplicación o desarrollar sobre ella.

En primer lugar, se debe garantizar de que el entorno de ejecución o desarrollo es el mismo que el que se usó en el momento del desarrollo de esta aplicación. Esto evitará posible errores entre conflictos de versiones de los paquete y la API usada.

Si se ha elegido instalar *Anaconda* es necesario seguir los pasos indicados en la documentación oficial, disponible en la dirección web:

<http://conda.pydata.org/docs/using/envs.html#use-environment-from-file>,

junto el fichero `conda-environment.yml` localizado en el directorio `/gui/configuration/` para replicar el entorno. Además, se ha facilitado el script `conda-create-environment.sh` localizado en el mismo directorio que muestra los pasos indicados en la documentación oficial. El script creará un entorno con los mismos paquetes y versiones al ser ejecutado.

Para una instalación manual de los módulos python en el anaconda se puede hacer lo siguiente:

- Para instalar el módulo `pyfits`:

1. Abrir una interfaz de línea de comandos (CLI).
2. Ejecutar el comando:

```
pip install pyfits
```

- Para instalar el módulo `PyQt4`:

1. Abrir una interfaz de línea de comandos (CLI).
2. Ejecutar el comando:

```
conda install pyqt
```

En segundo lugar, es necesario asegurarse que la ruta del programa `pyrcc4` esté en la variable `PATH` de nuestro sistema. Se ha facilitado un script, llamado `pyrcc4-check.py` localizado en `/gui/configuration/` para sistemas GNU/Linux, que comprueba la accesibilidad de este programa.

Por último, el fichero `pycharm50-settings.jar` localizado en `/gui/configuration/` agrupa todas las configuraciones del IDE *PyCharm Community Edition 5.0.3*, que podrán servir a los futuros desarrolladores que mantengan esta aplicación.

10.3.6 Distribuciones

El módulo *PyInstaller* se usa para generar las distribuciones de la aplicación. Una distribución de la aplicación contiene todos los archivos necesarios para usar la aplicación sin que sea necesario satisfacer los requisitos de la aplicación.

Para generar la distribución de la aplicación utilizando *PyInstaller* es necesario seguir los siguientes pasos:

1. Editar el fichero `run.py`, comentando las líneas mostradas a continuación y salvando los cambios realizados.

```
5 # Obtener la ruta absoluta
6 import os
7 path = os.getcwd()
8 # Borrar ficheros de recursos
9 from sources.resources.make import remove_all_resources
10 remove_all_resources(path=path)
11 # Generar ficheros fuentes de los recursos
12 from sources.resources.make import compile_all_resources
13 compile_all_resources(path=path)
14 # Volver al directorio de la aplicación
15 os.chdir(path)
```

2. Abrir una interfaz de línea de comandos (CLI).

3. Situar en el directorio raíz de la aplicación.
4. Generar la distribución ejecutando:


```
pyinstaller run.py
```
5. Una vez terminado el proceso de generación aparecerán dos directorios y un fichero a partir del directorio raíz de la aplicación: directorio `build/run`, directorio `dist/run` y fichero `run.spec`.
6. El ejecutable generado en el proceso `run` se encuentra en el directorio `dist/run`.

10.4 Descripción de la aplicación

En este apartado se mencionaran aspectos más técnicos que podrán ser de utilidad a posibles desarrolladores que deseen mantener esta aplicación en el futuro.

Los aspectos considerados más relevantes son: describir el árbol de directorios utilizado en el desarrollo, describir las clases que conforman el núcleo de la aplicación, describir los eventos que la aplicación utiliza y describir el formato de los ficheros usados como entrada y el formato de los ficheros generados.

10.4.1 Árbol de directorios

Conocer el árbol de directorios empleado en esta aplicación permite localizar los ficheros fuentes que la componen y otros elementos como documentación, distribuciones y configuraciones.

El directorio raíz de esta aplicación se encuentra en `/gui/` en el dispositivo de almacenamiento asociado a este documento. La figura10.4 ilustra el árbol de directorios de esta aplicación.

Los directorios más relevantes de la aplicación son los siguientes:

- **sources**: directorio donde se ubican los códigos fuente de esta implementación.
- **sources**: En este directorio se encuentran todos los códigos fuentes y los recursos utilizados por la aplicación. En el argot de los desarrolladores en python, se puede decir que este documento contiene los *paquetes* y *módulos* necesarios de la aplicación.

Estos paquetes son los siguientes directorios:

- **analyzer**: Paquete que contiene los módulos que analizan la factibilidad de las expresiones matemáticas sobre las columnas de los ficheros de datos.
- **constant**: Paquete que contiene los módulos que definen los valores constantes utilizados en la aplicación.
- **functionality**: Paquete que concentra los módulos que conforman el núcleo de la aplicación. Para conocer con mayor detalle las clases que implementadas en este módulo, el lector puede consultar la sección10.4.2.
- **legacy**: Paquete que concentra los *scripts* necesarios que ésta aplicación debía integrar. Estos han sido desarrollados inicialmente por *Edouard Bernard* y modificados por los autores de este documento, con la finalidad de que las funcionalidades contenidas en estos scripts estuvieran modularizadas en funciones independiente de la existencia de variables globales.
- **resources**: Este paquete contiene módulos generados y un conjunto de ficheros de diversa índole, todos ellos relacionados con el aspecto final de la aplicación.

Las extensiones de los ficheros ubicado en este directorio son:

- *ui*: ficheros con la definición los elementos visibles de la aplicación. La manipulación de estos ficheros se realiza con la herramienta Qt Designer¹. El formato de estos ficheros se define en esta dirección web:
<http://doc.qt.io/qt-4.8/designer-ui-file-format.html>.
- *qrc*: ficheros que define los recursos utilizados, actuando como mecanismo de recolección de recursos independiente de la plataforma. El uso de estos ficheros está descrito en esta dirección web:
<http://doc.qt.io/qt-4.8/resources.html>.

¹Para más información sobre Qt Designer consultar la dirección web: <http://doc.qt.io/qt-4.8/designer-manual.html>

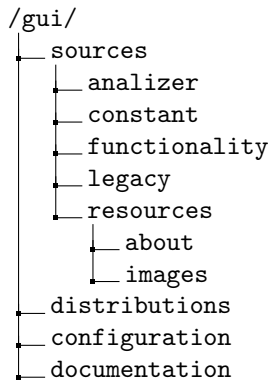


Figura 10.4: Árbol de directorios de la aplicación de escritorio con interfaz GUI.

- *py*: módulos python generados a partir de los ficheros *ui* y *qrc*.
- *png*: imágenes de los botones y acciones.
- *html*, *css*, *js*: ficheros que definen la ventana de información de esta aplicación. Estos ficheros están almacenados en *resources/about*.
- **distributions**: En este directorios se localizan las distribuciones de la aplicación compiladas, El lector puede consultar la sección 10.3.6, para conocer más acerca del proceso para su generación y la sección 10.5.1 para conocer cómo ejecutarlas.
- **configuration**: En este directorio están contenidos los ficheros de configuración del entorno de desarrollo usado para crear la aplicación.
- **documentation**: Directorio que aloja algunos ficheros que puede servir a posibles futuros desarrolladores como los diagramas UML de las principales clases de esta aplicación. Se utilizó para la creación de estos diagramas la aplicación Dia².

Los ficheros y directorios de configuración Git para este proyecto se encuentran ocultos en las siguientes rutas:

```

/cli/cpu/.git y
/cli/cpu/.gitignore.

```

10.4.2 Descripción de clases

En esta sección se describen grosso modo las clases que conforman el núcleo principal de la aplicación.

10.4.2.1 MainWindow

Esta clase implementa la ventana principal de la aplicación. Se encarga de instanciar los elementos, gestionar la mayoría de eventos y definir casi todas las acciones de la aplicación. La figura 10.5 ilustra esta clase en un diagrama UML. Debido a la gran cantidad de atributos y métodos que contiene esta clase, sólo se han mencionado las más relevantes en dicha figura.

²Dia Diagram Editor proyecto con licencia GPLv2 disponibles en la dirección web <http://dia-installer.de/>



Figura 10.5: Diagrama de la clase `MainWindow` con notación UML.

10.4.2.2 ManageDiagram

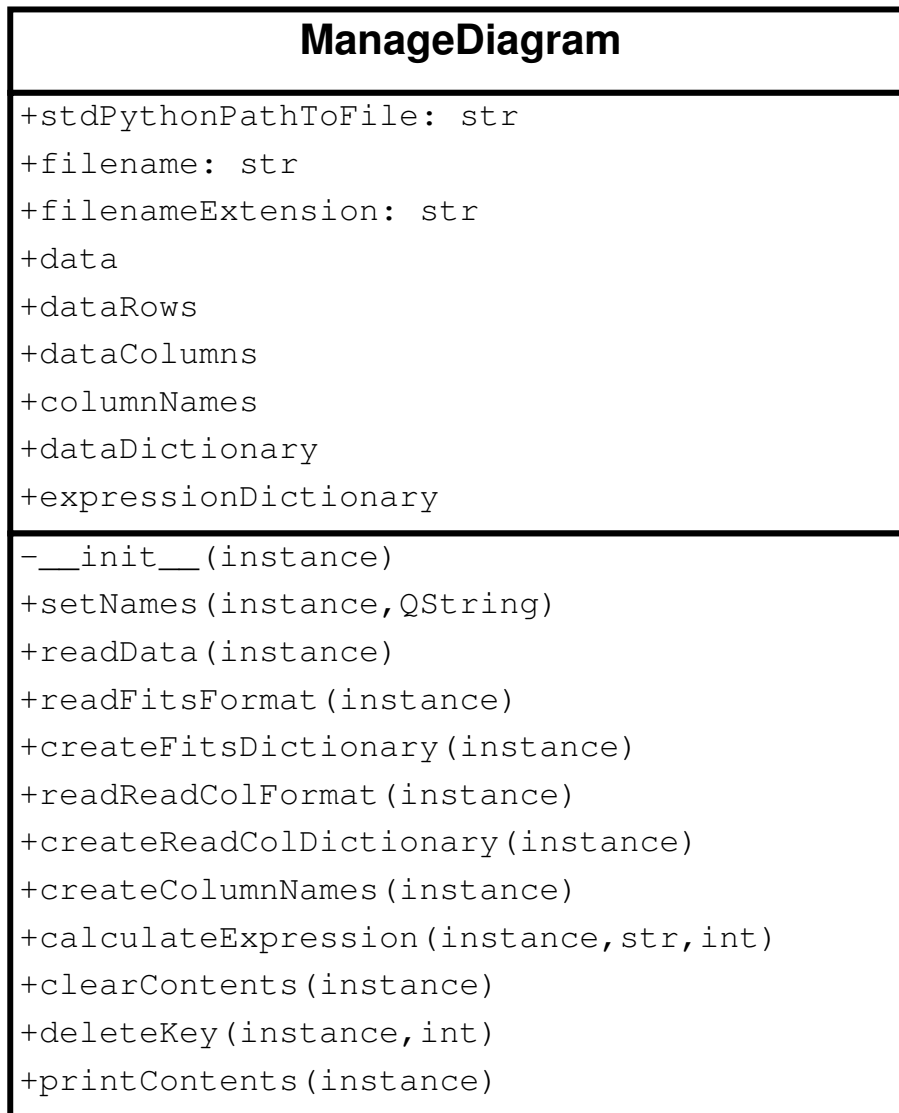


Figura 10.6: Diagrama de la clase `ManageDiagram` con notación UML.

La figura 10.6 muestra el diagrama UML de la clase `ManageDiagram`. Esta clase se encarga de manipular el fichero de datos del diagrama y calcular las expresiones matemáticas asociadas al diagrama.

10.4.2.3 Diagram



Figura 10.7: Diagrama de la clase `Diagram` con notación UML.

Esta clase representa cada uno de los diagramas color-magnitud mostrados en la aplicación. La figura 10.7 ilustra esta clase, usando la representación UML. Existen varios atributos y métodos propios de la clase y no de los objetos instanciados, también llamados atributos y métodos estáticos. Estos métodos y atributos están subrayados en el diagrama UML de la figura 10.7 y son los que permiten replicar las acciones hechas sobre un diagrama en el resto de diagramas. Esta clase tiene una estrecha relación con la clase `Bundle`, para poder resaltar los puntos de un bundle seleccionado, añadir nuevos puntos a los bundle con eventos de ratón sobre el diagrama, etcétera.

10.4.2.4 Bundle

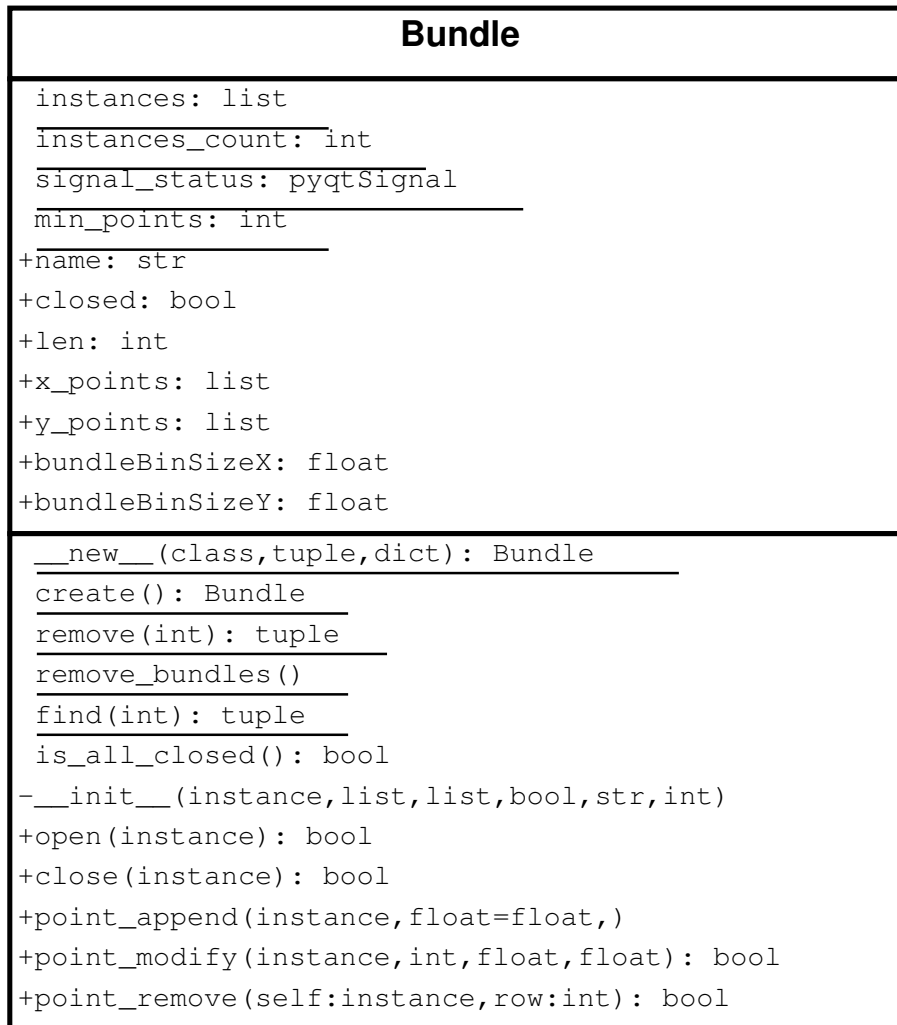


Figura 10.8: Diagrama de la clase `Bundle` con notación UML.

Esta clase es la que implementa los bundles o estructuras poligonales sobre el diagrama color-magnitud. La figura 10.8 ilustra esta clase con un diagrama de clase UML. Se puede apreciar, al igual que la clase *Diagram*, que ésta clase tiene atributos y métodos estáticos. La mayoría de ellos permiten realizar acciones sobre los diagramas y gestionar adecuadamente otras acciones como el cambio de selección de un bundle a otro, la selección de cualquier punto en el bundle activo, etcétera. Por estos motivos, es notorio que esta clase debe guardar una estrecha relación con la clase *Diagram*.

10.4.2.5 Histogram

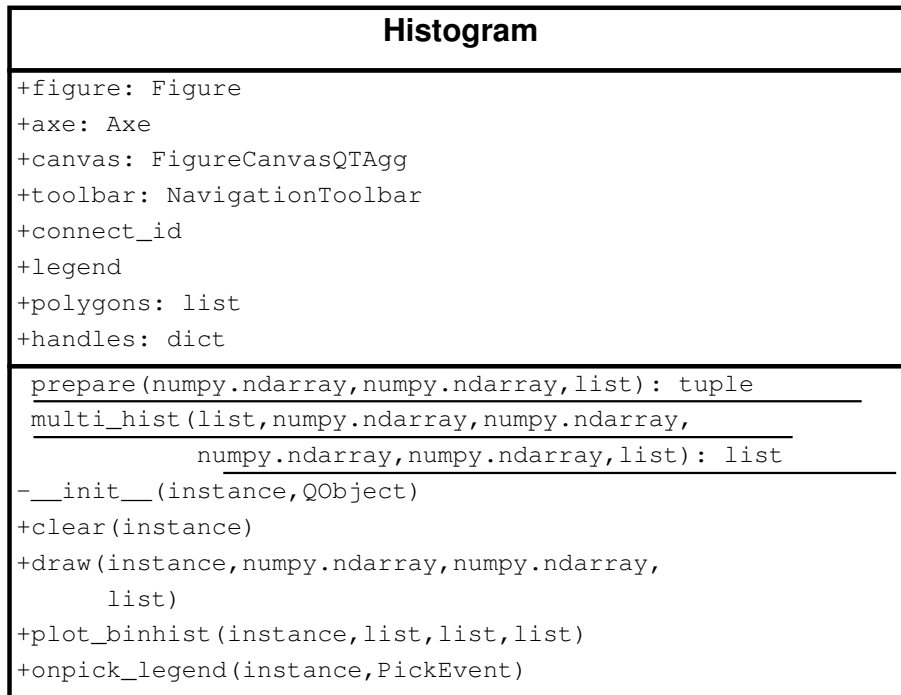


Figura 10.9: Diagrama de la clase `Histogram` con notación UML.

Esta clase implementa la integración de los histogramas en la aplicación. Fue inspirada en los scripts desarrollados por *Edouard Bernard*. Esta clase también dispone de métodos estáticos que permiten actualizar la información de los histogramas relacionados con cada uno de los diagramas color-magnitud.

La figura 10.9 la ilustra.

10.4.2.6 NavigationToolbar

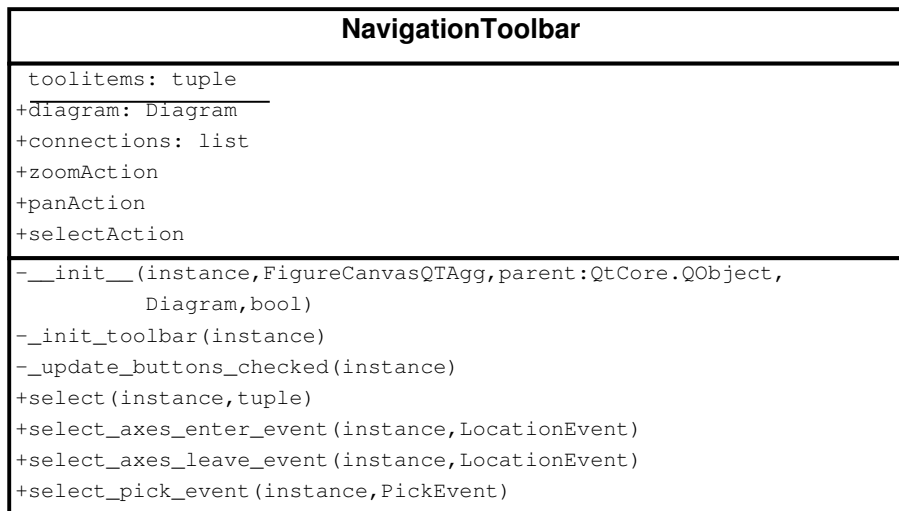


Figura 10.10: Diagrama de la clase `NavigationToolbar` con notación UML.

Esta clase es una modificación de la barra de herramientas asociada a un diagrama de la librería `matplotlib` para PyQt. Esta clase se ilustra en la figura 10.10. Existen dos clases muy similares entre sí, ubicadas en el directorio `/gui/sources/functionality/` en los módulos: `plotting.py` y `histogram.py`.

La primera de ellas, localizada en `/gui/sources/functionality/plotting.py`, modificó la clase base para permitir al usuario definir un punto perteneciente a otro bundle en el bundle actual. Esto evitará posibles problemas con la sensibilidad del ratón y la precisión del usuario para hacer clic dos veces en un mismo punto.

La segunda, localizada en `/gui/sources/functionality/histogram.py`, modificó la clase base para evitar problemas de redefinición del diagrama ofrecido por defecto en la clase base.

10.4.3 Eventos de la aplicación

A continuación se explicaran los eventos principales que se producen en la aplicación y que hacen posible que las funcionalidades requeridas estén operativas.

El primero de estos eventos es la habilitación de la pestaña *Showing Diagrams*. La habilitación de esta pestaña se produce al cargar los siguientes elementos en la pestaña *Load Diagrams and Data*, por este orden:

1. Los ficheros asociados a los diagramas observado y modelo.
2. Las expresiones X e Y en los parámetros de entrada de ambos diagramas.

Las ventanas flotantes *Diagram tool*, *Bundle tool* e *Histogram tool* estarán habilitadas siempre que esté seleccionada la pestaña *Showing Diagrams*. Además de esto para que se habilite la ventana flotante *Histogram tool* es necesario:

- Tener definido al menos un bundle cerrado con sus valores de rejilla (*Bundle Bin Size*) mayores que cero absoluto.

Por último comentar el evento que produce la habilitación de las líneas de texto donde introduciremos los rangos de edad y metalicidad (pestaña *Counting Stars*).

- Para habilitar la línea de texto de rangos de edad tendremos que haber cargado la expresión *Age* en el diagrama modelo en la pestaña *Load Diagrams and Data*.
- Para habilitar la línea de texto de rangos de metalicidad tendremos que haber cargado la expresión *Metallicity* en el diagrama modelo en la pestaña *Load Diagrams and Data*.

10.4.4 Formato de los ficheros

En este apartado se describen todos los formatos de ficheros de entrada y salida que usa la aplicación.

La ficheros de entrada contienen la información relativa a las estrellas y extrayendo la información de la forma adecuada podremos hacer uso de la aplicación.

Por otra parte los ficheros de salida son archivos que el usuario puede generar en el momento que considere oportuno.

10.4.4.1 Ficheros de parámetros

La aplicación maneja dos formatos de ficheros de parámetros. Las extensiones de estos dos tipos de ficheros son `*.in` y `*.ini`.

Los ficheros con extensión `.ini` son ficheros que sirven para salvar y restaurar el estado de la aplicación. Este tipo de ficheros tiene un formato estándar³. La clase `QSettings`, que permite almacenar una instantánea de la aplicación que puede ser restaurada más adelante. Un posible ejemplo de este tipo de ficheros es:

```

1  [Windows]
2  geometry=@Rect(0 24 1366 744)
3
4  [Observed]
5  obsFile=./files/m31_warp.abs.fits
6  obsExpXaxis=c4-c6
7  obsExpYaxis=c6
8  obsLabXaxis=ObsX
9  obsLabYaxis=ObsY
10
11 [Model]
12 modFile=./files/basti_50b_15000_10M.disp.fits
13 modExpXaxis=c1-c2
14 modExpYaxis=c2
15 modExpAge=c3
16 modExpMetal=c4
17 modLabXaxis=ModX
18 modLabYaxis=ModY
19
20 [Bundles]
21 count=2
22 bundle\size=2
23 bundle\1\closed=true
24 bundle\1\len=3
25 bundle\1\binSizeX=0
26 bundle\1\binSizeY=0
27 bundle\1\point\1\X=-0.25793
28 bundle\1\point\1\Y=-2.02164
29 bundle\1\point\2\X=-0.13474
30 bundle\1\point\2\Y=4.26517
31 bundle\1\point\3\X=1.2819
32 bundle\1\point\3\Y=3.47624
33 bundle\1\point\4\X=-0.25793
34 bundle\1\point\4\Y=-2.02164
35 bundle\1\point\size=4
36 bundle\2\closed=true
37 bundle\2\len=3
38 bundle\2\binSizeX=0
39 bundle\2\binSizeY=0

```

³Para obtener más información sobre el formato `.ini` consultar en la dirección web: https://en.wikipedia.org/wiki/INI_file

```

40 bundle\2\point\1\X=0.94754
41 bundle\2\point\1\Y=-3.57485
42 bundle\2\point\size=4
43 bundle\2\point\2\X=0.22886
44 bundle\2\point\2\Y=-0.28358
45 bundle\2\point\3\X=0.98478
46 bundle\2\point\3\Y=2.4154
47 bundle\2\point\4\X=0.94754
48 bundle\2\point\4\Y=-3.57485
49
50 [Tables]
51 age=
52 metallicity=
53
54 [Others]
55 refineGrid=false
56 intMother=
57 areaReg=
58 binningOffsets=
59 numberIterations=
60 numberProcessors=

```

Los ficheros con extensión `.in` tienen el mismo propósito que los ficheros `.ini`. Sin embargo, el formato de este tipo de ficheros fue establecido por el personal del Instituto Astrofísico de Canarias. Además estos ficheros servirán como entrada a otro de los *scripts* realizados por *Edouard Bernard* que finalmente no fue incorporado a la aplicación. Un posible ejemplo de este tipo de ficheros es:

```

1  # Observed CMD
2  obscmd = 'C:\io\m31_warp.abs.fits'
3  obsExpXaxis = c4-c6
4  obsExpYaxis = c6
5
6  # Model CMD
7  modcmd = 'C:\io\basti_50b_15000_10M.disp.fits'
8  modExpXaxis = c1-c2
9  modExpYaxis = c2
10 modExpAge = c3
11 modExpMetallicity = c4
12
13 # CMD color and magnitude ranges
14 ran_c = [-0.49, 2.49]
15 ran_m = [4.99, -4.99]
16
17 # Bundles (one line per bundle: b=[[x_vertices], [y_vertices]])
18 b = [[-0.25, -0.0, 1.04, -0.05, -0.25], [-2.0, 3.91, 3.45, -2.0, -2.0]]
19 b = [[0.6, 0.6, 0.9, 0.6], [0.5, 1.25, 0.5, 0.5]]
20 b = [[0.89, 0.93, 1.3, 1.6, 0.89], [0.5, -0.36, -1.5, 0.5, 0.5]]
21 b = [[0.75, 2.3, 0.75], [-4.0, -4.0, -4.0]]
22
23 # Bin sizes in color and magnitude (one line per bundle)
24 bin_cm = [0.04, 0.03]
25 bin_cm = [0.1, 0.5]
26 bin_cm = [0.3, 0.5]
27 bin_cm = [0.1, 0.15]
28
29 # Age (in Gyr) and metallicity (in Z) bins
30 age = [0.0, 0.25, 0.5, 0.75, 1.0]
31 met = [0.0004, 0.0007, 0.0011, 0.0016, 0.0022]
32

```

```

33 # Parameters to calculate met bins
34 number_of_bins = 12
35 z_min = 0.0001
36 z_max = 0.002
37 z_sun = 0.0198
38
39 # Normalizing constants (intmother in  $M_{\text{sun}}$ ; areareg in  $\text{pc}^2$ )
40 intmother = 1.
41 areareg = 1.
42
43 # Offset observed CMD in color & magnitude (i.e. ~ reddening & distance)...
44 # (S)earch minimum, check only (O)ne position, or sample full (G)rid?
45 dist_redd = 'S'
46
47 # ...centered on:
48 dist_redd_center = [0.0, 0.0]
49
50 # Refine the grid to find the minimum in the chi2 map?
51 refine_grid = False
52
53 # Number of binning offsets in color and magnitude
54 shifts = (2, 2)
55
56 # Also shift the binning in age?
57 refine_age = False
58
59 # Number of iterations of Poissonian resampling of the observed CMD
60 nPoisson = 4
61
62 # Number of processors to use when running in parallel
63 nProcessors = mp.cpu_count()

```

10.4.4.2 Ficheros de datos

La aplicación está preparada para manejar dos tipos de ficheros de datos: ficheros de texto plano y ficheros *FITS*.

El formato en texto plano tiene la misma estructura que la indicada en la sección 1.2. En este formato cada estrella representa una línea y las columnas representan los atributos de dicha estrella.

El formato *FITS* es un formato extendido en el ámbito astronómico para el almacenamiento de datos. En este formato la información de los datos almacenados se definen mediante una cabecera. Para poder extraer los datos de este tipo de ficheros es necesario el paquete *pyfits*.

10.5 Manual de usuario

Esta sección se dedicada a conocer la aplicación de escritorio con interfaz gráfica desde el punto de vista de un usuario. En concreto se hablará de cómo ejecutarla, la ubicación de los elementos, las posibles acciones disponibles para el usuario, etcétera.

A lo largo de esta sección se remitirá al lector a secciones en las que éste puede profundizar sobre diferentes aspectos ya descritos, si así lo desea.

10.5.1 Modos de ejecución

Las personas que deseen hacer uso de esta aplicación deben saber que existen dos modos de ejecutarla:

10.5.1.1 Usando el código fuente

La localización del código fuente de esta aplicación se encuentra en el directorio `/gui/` dentro del dispositivo de almacenamiento adjunto a este documento. Alternativamente, se puede encontrar una copia comprimida en directorio `/compressed/gui/`.

Este modo permite la ejecución en múltiples entornos, siempre que se satisfagan los requisitos de la sección 10.3.1.

Los elementos necesarios para esta modo de ejecución de la aplicación son: el intérprete de Python y una interfaz de línea de comandos (CLI).

Los pasos que se deben seguir para ejecutar la aplicación desde el código fuente son:

1. Abrir una interfaz de línea de comandos (CLI).
2. Cambiar al directorio donde se encuentra el código fuente.
3. Asegurarse de disponer de los privilegios necesarios sobre los fuentes y directorios para la ejecución.
4. Lanzar la ejecución del módulo principal con el siguiente comando:

```
python run.py
```

Alternativamente, se puede utilizar el fichero comprimido del código fuente, descomprimirlo en la ubicación deseada y seguir los pasos anteriores.

10.5.1.2 Eligiendo algunas de las distribuciones ofrecidas

La localización de las distribuciones ofrecidas se encuentran en el directorio `/gui/distributions/` dentro del dispositivo de almacenamiento adjunto a este documento.

Este modo sólo permite la ejecución de entornos muy similares al de la distribución elegida. Las distribuciones ofrecidas son:

- Windows 7 Professional para 64 bits.
- Kubuntu 14.04 LTS para 64 bits.

No se requiere software adicional, debido a que las distribuciones son ficheros compilados que contienen todo lo necesario para la ejecución.

Si se desea conocer más sobre cómo generar una distribución compilada, dirigirse a la sección 10.3.6.

10.5.2 Visión general

Una vez que se conoce las formas de ejecutar la aplicación, es el turno de explicar los diferentes elementos que componen la aplicación. Es destacable que el idioma de la aplicación es el inglés.

A continuación, se detallan las acciones principales y la disposición general de los elementos principales de la aplicación.

10.5.2.1 Acciones principales

Las acciones principales de la aplicación son las siguientes:

10.5.2.1.1 Open settings Esta acción permite cargar desde un fichero con formato estándar una sesión previamente salvada. El formato de este fichero se explica en la sección 10.4.4.1. Al seleccionar esta acción se despliega una ventana emergente que permite navegar por el árbol de directorios del sistema operativo y filtrar los archivos por la extensión *.ini*.



Figura 10.11: Icono asociado con la acción *Open settings*.

Ejecutar esta acción es posible pulsando la combinación de teclas **Ctrl+O** o haciendo clic en el icono de la figura 10.11.

10.5.2.1.2 Save settings Esta acción permite salvar la sesión en un fichero con formato estándar, para que posteriormente pueda ser restaurada. El formato de este fichero se explica en la sección 10.4.4.1. Cuando se selecciona esta acción se abre un cuadro de diálogo que filtra los archivos por la extensión *.ini*.



Figura 10.12: Icono asociado con la acción *Save settings*.

Ejecutar esta acción es posible pulsando la combinación de teclas **Ctrl+S** o haciendo clic en el icono de la figura 10.12.

10.5.2.1.3 Open parameters Esta acción permite cargar la sesión previamente salvada desde un fichero de parámetros. El formato de este fichero se explica en la sección 10.4.4.1. Al hacer uso de esta acción se abre un cuadro de diálogo que filtra los archivos por la extensión *.in*.



Figura 10.13: Icono asociado con la acción *Open parameter*.

Ejecutar esta acción es posible pulsando la combinación de teclas **Ctrl+L** o haciendo clic en el icono de la figura 10.13.

10.5.2.1.4 Save parameters Esta acción permite salvar la sesión en un fichero de parámetros, para que posteriormente pueda ser restaurada. El formato de este fichero se explica en la sección 10.4.4.1. Al seleccionar esta acción se desplegará un cuadro de diálogo que filtra los archivos por la extensión *.in*.



Figura 10.14: Icono asociado con la acción *Save parameters*.

Ejecutar esta acción es posible pulsando la combinación de teclas **Ctrl+W** o haciendo clic en el icono de la figura 10.14.

10.5.2.1.5 Open observed diagram Esta acción permite cargar los datos de las estrellas que se utilizarán en el diagrama observado. Los formatos permitidos son los descritos en la sección 10.4.4.2. Al ejecutar esta acción se abre un cuadro de diálogo que filtra o no los ficheros por la extensión *.fits*.



Figura 10.15: Icono asociado con la acción *Open observed diagram*.

Ejecutar esta acción es posible pulsando la combinación de teclas **Ctrl+1** o haciendo clic en el icono de la figura 10.15.

10.5.2.1.6 Open model diagram Esta acción permite cargar los datos de las estrellas que se utilizarán en el diagrama modelo. Los formatos permitidos son los descritos en la sección 10.4.4.2. Al ejecutar esta acción se abre un cuadro de diálogo que filtra o no los ficheros por la extensión *.fits*.



Figura 10.16: Icono asociado con la acción *Open model diagram*.

Ejecutar esta acción es posible pulsando la combinación de teclas **Ctrl+2** o haciendo clic en el icono de la figura 10.16.

10.5.2.1.7 Show/Hide diagram tool Esta acción permite mostrar u ocultar las herramientas para configurar la vista de los diagramas.

Esta acción está vinculada a una de las *ventanas flotantes* y puede encontrarse en el estado de habilitada o deshabilitada en función de si los parámetros de expresión de datos de los diagramas fueron o no cargados correctamente.



Figura 10.17: Icono asociado con la acción *Open model diagram*.

Ejecutar esta acción es posible pulsando la combinación de teclas **Ctrl+D** o haciendo clic en el icono de la figura 10.17.

10.5.2.1.8 Show/Hide bundle tool Esta acción que muestra u oculta la herramienta que permite manipular los bundles en los diagramas.

Esta acción está vinculada a una *ventanas flotante* y su estado puede ser habilitada o deshabilitada en función de la correcta lectura de los parámetros de expresión de datos de los diagramas.



Figura 10.18: Icono asociado con la acción *Show/Hide bundle tool*.

Ejecutar esta acción es posible pulsando la combinación de teclas **Ctrl+B** o haciendo clic en el icono de la figura 10.18.

10.5.2.1.9 Show/Hide histogram tool Esta acción que muestra u oculta la herramienta que permite visualizar el histograma de los bundles cerrados.

Esta acción está vinculada a una *ventanas flotante* y su estado puede ser habilitada o deshabilitada en función la existencia de *bundles* cerrados con valores superiores al cero absoluto en los parámetros *bin size*.



Figura 10.19: Icono asociado con la acción *Show/Hide histogram tool*.

Ejecutar esta acción es posible pulsando la combinación de teclas **Ctrl+H** o haciendo clic en el icono de la figura 10.19.

10.5.2.1.10 Reset original view Esta acción permite restaurar la vista original del diagrama asociado.



Figura 10.20: Icono asociado con la acción *Reset original view*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.20.

10.5.2.1.11 Back to previous view Esta acción permite volver a una vista definida previamente a la vista actual en el diagrama asociado.



Figura 10.21: Icono asociado con la acción *Back to previous view*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.21.

10.5.2.1.12 Forward to next view Esta acción permite restaurar una vista definida posteriormente a la vista actual en el diagrama asociado.



Figura 10.22: Icono asociado con la acción *Forward to next view*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.22.

10.5.2.1.13 Pan axes with left mouse, zoom with right Esta acción si se usa dentro del diagrama asociado con el clic izquierdo del ratón permite desplazar su contenido y con el click derecho permite hacer zoom.



Figura 10.23: Icono asociado con la acción *Pan axes with left mouse, zoom with right*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.23.

10.5.2.1.14 Zoom to rectangle Esta acción permite hacer zoom sobre el diagrama asociado.



Figura 10.24: Icono asociado con la acción *Zoom to rectangle*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.24.

10.5.2.1.15 Select a point Esta acción permite copiar un punto definido de otro bundle al bundle actual. Es necesario que el usuario se asegure que las acciones *Pan axes with left mouse*, *zoom with right* y *Zoom to rectangle* estén desactivadas antes de activar esta acción. De lo contrario, esta acción no funcionará adecuadamente.



Figura 10.25: Icono asociado con la acción *Select a point*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.25.

10.5.2.1.16 Save the figure Esta acción permite guardar como imagen en un fichero la vista del diagrama asociado.



Figura 10.26: Icono asociado con la acción *Save the figure*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.26.

10.5.2.1.17 Create bundle Esta acción permite crear un nuevo bundle sin puntos y con los valores de su rejilla en cero absoluto. El bundle creado pasa a ser el bundle seleccionado.



Figura 10.27: Icono asociado con la acción *Create bundle*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.27.

10.5.2.1.18 Open bundle Esta acción permite abrir el bundle seleccionado, lo que permitirá definir nuevos puntos en el bundle.

El estado está deshabilitado mientras el bundle seleccionado esté abierto y está habilitado si el bundle seleccionado está cerrado.



Figura 10.28: Icono asociado con la acción *Open bundle*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.28.

10.5.2.1.19 Close bundle Esta acción permite cerrar el bundle seleccionado, imposibilitando definir nuevos puntos en el bundle.

El estado está deshabilitado mientras el bundle seleccionado esté cerrado y está habilitado si el bundle seleccionado está abierto.



Figura 10.29: Icono asociado con la acción *Close bundle*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.29.

10.5.2.1.20 Remove bundle Esta acción permite eliminar el bundle seleccionado.

El estado de esta acción está deshabilitado mientras no existan bundles y habilitado en otro caso.



Figura 10.30: Icono asociado con la acción *Remove bundle*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.30.

10.5.2.1.21 Create point Esta acción permite añadir un punto definido por el valor de sus coordenadas al conjunto de puntos que conforman el bundle seleccionado.

Por defecto el estado de esta acción es deshabilitado.

Para habilitarla, se debe seleccionar algún bundle abierto y hacer clic derecho sobre alguno de los diagramas. Esto rellenará los valores de las coordenadas del punto del diagrama para permitir añadirlo al bundle seleccionado.



Figura 10.31: Icono asociado con la acción .

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.31.

10.5.2.1.22 Modify point Esta acción permite modificar un punto ya definido editando el valor de sus coordenadas sobre el bundle seleccionado.

De forma predeterminada esta acción estará deshabilitada. Para habilitarla es necesario hacer clic la tabla de valores de los puntos definidos del bundle actual. Esto rellenará los valores de las coordenadas de dicho punto para permitir modificarlas.



Figura 10.32: Icono asociado con la acción *Modify point*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.32.

10.5.2.1.23 Remove point Esta acción permite eliminar un punto definido del bundle seleccionado.

Por defecto, esta acción permanecerá deshabilitada, hasta que se seleccione algún punto de los definidos en el bundle seleccionado, localizado en la tabla de valores de coordenadas.



Figura 10.33: Icono asociado con la acción *Remove point*.

Ejecutar esta acción es posible haciendo clic en el icono de la figura 10.33.

10.5.2.1.24 About this Esta acción muestra en una ventana emergente la información sobre la aplicación y una breve descripción sobre las personas que tuvieron alguna relación con la implementación de esta aplicación.



Figura 10.34: Icono asociado con la acción *About this*.

Ejecutar esta acción es posible pulsando la combinación de teclas **F1** o haciendo clic en el icono de la figura 10.34.

10.5.2.1.25 About Qt Esta acción muestra en una ventana emergente la información sobre el *framework base* de desarrollo de la aplicación.



Figura 10.35: Icono asociado con la acción *About Qt*.

Ejecutar esta acción es posible pulsando la combinación de teclas **Ctrl+F1** o haciendo clic en el icono de la figura 10.35.

10.5.2.1.26 Exit application Esta acción hará que la aplicación se cierre. Es equivalente a usar el icono de cierre de ventanas del sistema operativo.



Figura 10.36: Icono asociado con la acción *Exit application*.

Ejecutar esta acción es posible pulsando la combinación de teclas **Ctrl+Q** o haciendo clic en el icono de la figura 10.36.

10.5.2.2 Disposición general

La figura 10.37 determina la disposición de los elementos de principales de la aplicación desarrollada.

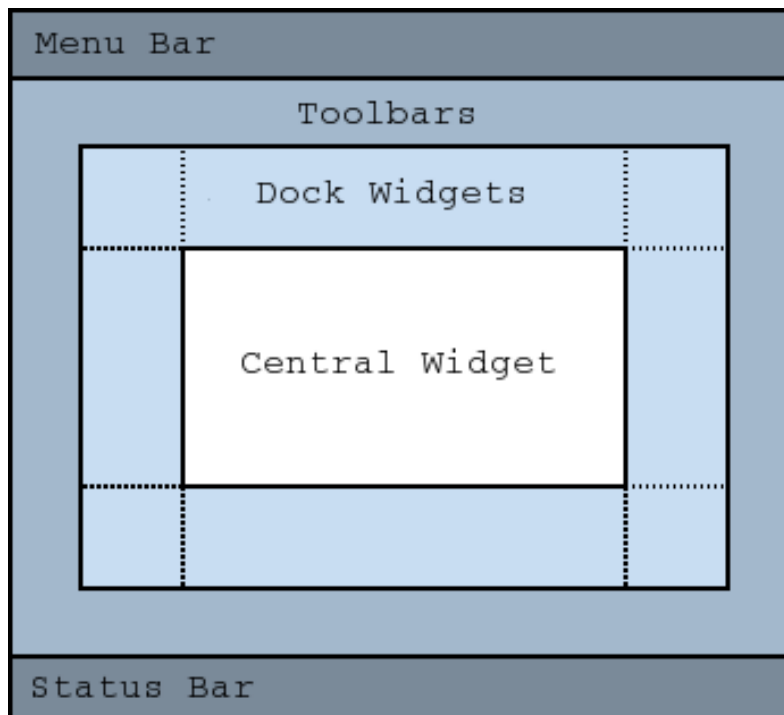


Figura 10.37: Disposición general de los elementos de la aplicación.





10.5.2.2.1 Menu Bar La barra de menú presenta las diferentes opciones disponibles en la aplicación agrupadas en categorías, llamadas menú. Los menús o categorías disponibles en esta aplicación son las siguientes:

■ File

Este menú agrupa las acciones relacionadas con los ficheros y permite salir de la aplicación. Las opciones relativas a los ficheros son: cargar datos y salvar cambios.



● Open


Submenú que concentra las acciones de cargar datos y abrir las sesiones almacenadas. Estas acciones son:

- *Parameters* : se trata de la acción *Open parameters*.
- *Observed* : se trata de la acción *Open observed diagram*.
- *Model* : se trata de la acción *Open model diagram*.
- *Settings* : se trata de la acción *Open settings*.

● Save




Submenú que concentra las operaciones de salvar los cambios hechos en la aplicación. Estas operaciones son:

- *Parameters* : se trata de la acción *Save parameters*.
- *Settings* : se trata de la acción *Save settings*.

- *Exit* : se trata de la acción *Exit application*.



■ Tools

Este submenú agrupa las acciones relacionadas con las herramientas que se utilizan para ajustar algunos parámetros de los dos diagramas color-magnitud: el observado y el modelo.

- *Diagram* : se trata de la acción *Show/Hide diagram tool*.
- *Bundle* : se trata de la acción *Show/Hide bundle tool*.
- *Histogram* : se trata de la acción *Show/Hide histogram tool*.










■ Help

Este menú agrupa las acciones que permiten obtener más información sobre la aplicación.

- *About* : se trata de la acción *About this*.
- *About Qt* : se trata de la acción *About Qt*.

10.5.2.2.2 Status Bar Esta aplicación no dispone de barra de estado.

10.5.2.2.3 Toolbar La barra de herramientas concentra las acciones más importantes que se encuentran la barra de menú. Éstas acciones son las siguientes:

- *Open settings* : se trata de la acción *Open settings*.
- *Save settings* : se trata de la acción *Save settings*.
- *Open parameters* : se trata de la acción *Open parameters*.
- *Save parameters* : se trata de la acción *Save parameters*.
- *Open observed diagram* : se trata de la acción *Open observed diagram*.
- *Open model diagram* : se trata de la acción *Open model diagram*.
- *Show/Hide diagram tool* : se trata de la acción *Show/Hide diagram tool*.
- *Show/Hide bundle tool* : se trata de la acción *Show/Hide bundle tool*.
- *Show/Hide histogram tool* : se trata de la acción *Show/Hide histogram tool*.

10.5.2.2.4 Ventana central Este elemento es el principal de la aplicación ya que concentra el área de acción principal del usuario. Este elemento está dividido en tres secciones, llamadas pestañas, que dividen las operaciones que el usuario puede realizar en función a su naturaleza.

Load Diagrams and Data En esta pestaña, representada en la figura 10.38, el usuario puede definir los ficheros de datos utilizados, las expresiones que determinan los datos para los diagramas observado y modelo y las etiquetas de los diagramas.

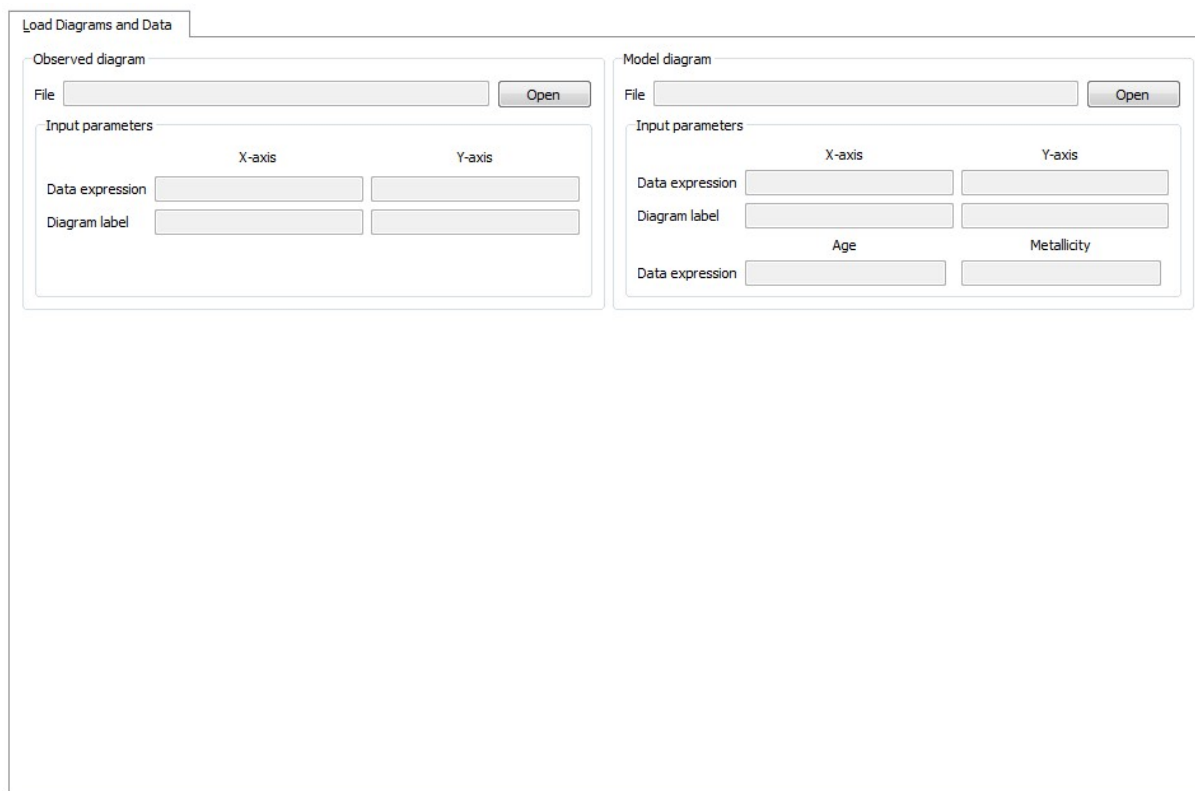


Figura 10.38: Presentación de la pestaña Load Diagrams and Data

Esta pestaña se subdivide en dos secciones: una sección dedicada a los parámetros del fichero relacionado con diagrama observado y otra sección dedicada a los parámetros del fichero relacionado con el diagrama modelo. Estas subsecciones están enmarcadas con un nombre identificativo.

Los parámetros que comparten ambas subsecciones son: ruta del fichero, expresiones matemáticas y etiquetas.

El fichero determina el origen de los datos que se usarán en cada diagrama color-magnitud. El usuario puede establecer estos ficheros, mediante las acciones *Open observed diagram* y *Open model diagram*.

Las expresiones matemáticas se definen sobre las columnas de los ficheros. El diagrama observado necesita dos expresiones para el diagrama color-magnitud. El diagrama modelo además de las dos anteriores tiene expresiones para edad y metalicidad. También se pueden definir etiquetas para los ejes de los diagramas color-magnitud.

Cada vez que cargamos una expresión en cualquiera de los diagramas, la aplicación mostrará si la carga resultó exitosa o no. Para ello la caja de texto de la expresión se colorea de la siguiente forma:

- **Color rojo:** la expresión introducida es incorrecta y no se ha cargado. Se informa al usuario de que se ha producido un error mediante un cuadro de diálogo que muestra un mensaje explicativo.
- **Color verde:** la expresión introducida es correcta y se ha cargado.
- **Color blanco:** no se ha introducido expresión.

Acceder a esta pestaña se puede hacer mediante la pulsación de teclas **Alt+L**.

El usuario puede consultar la sección 10.5.3.1 para obtener más detalles sobre el uso de esta pestaña.

Showing Diagrams En esta pestaña, figura 10.39, se muestran los diagramas color-magnitud de los diagramas observado y modelo.

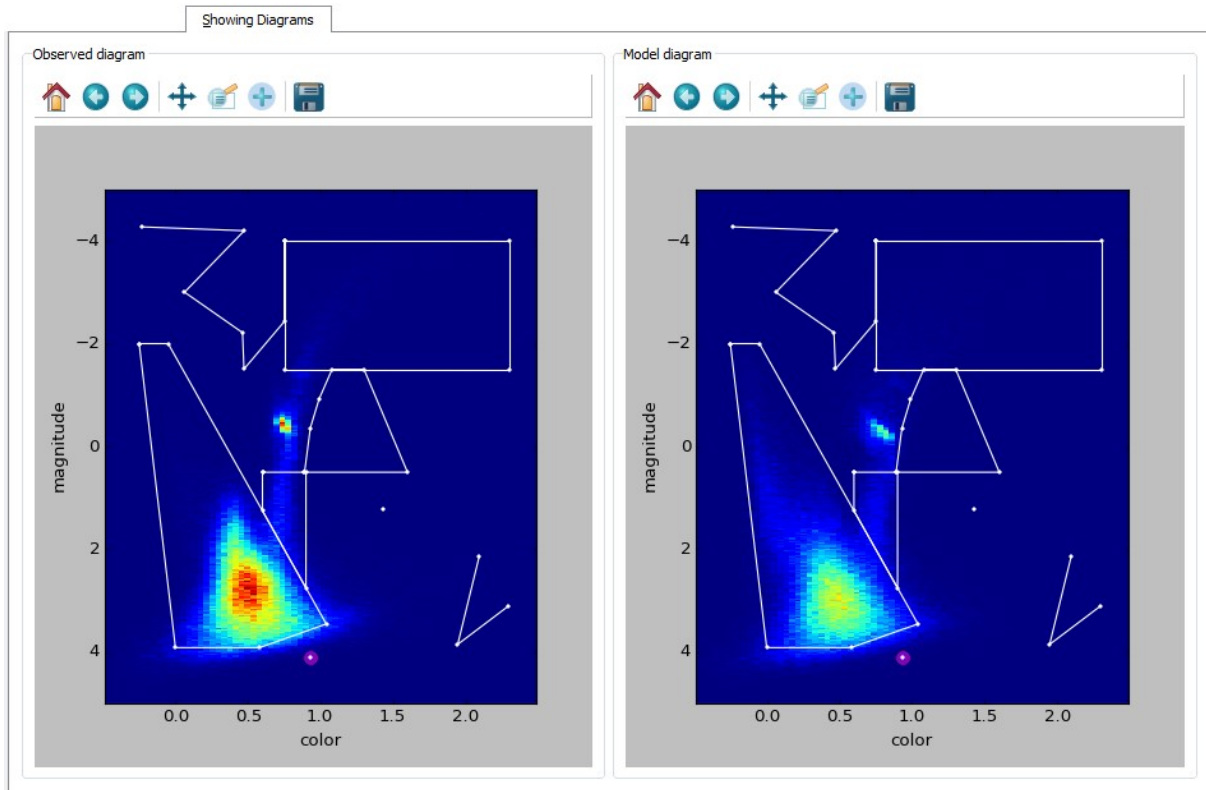


Figura 10.39: Presentación de la pestaña Showing Diagrams

Cada diagrama color-magnitud tiene una barra de herramientas que ofrece las siguientes acciones *Reset original view*, *Back to previous view*, *Forward to next view*, *Pan axes with left mouse*, *zoom with right*, *Zoom to rectangle*, *Select a point* y *Save the figure*.

A su vez en esta pestaña se habilita el acceso a las tres *ventanas flotantes* mediante la activación de las acciones *Show/Hide diagram tool*, *Show/Hide bundle tool* y *Show/Hide histogram tool*.

Cada vez que se añade un punto al bundle seleccionado, dicho punto se añadirá a ambos diagramas, independientemente del diagrama sobre el que se haya realizado el clic.

Acceder a esta pestaña se puede hacer mediante la pulsación de teclas **Alt+S**.

Para profundizar en el uso de esta pestaña el usuario puede consultar el apartado 10.5.3, comenzando por la sección 10.5.3.2 para finalizar en la sección 10.5.3.10.

Counting Stars Esta pestaña, figura 10.40, permite al usuario introducir una serie de parámetros.

Los siguientes parámetros son introducidos a través de líneas de texto y sus valores no se usan en ninguno de los cálculos que realiza la aplicación:

- **Int-Mother** (M_{sun}): texto.
- **Area-Reg** (pc^2): texto.
- **Binning Offsets X**: texto.
- **Binning Offsets Y**: texto.
- **Refine Age**: booleano determinado mediante una casilla de verificación.
- **Number of Iterations**: texto.

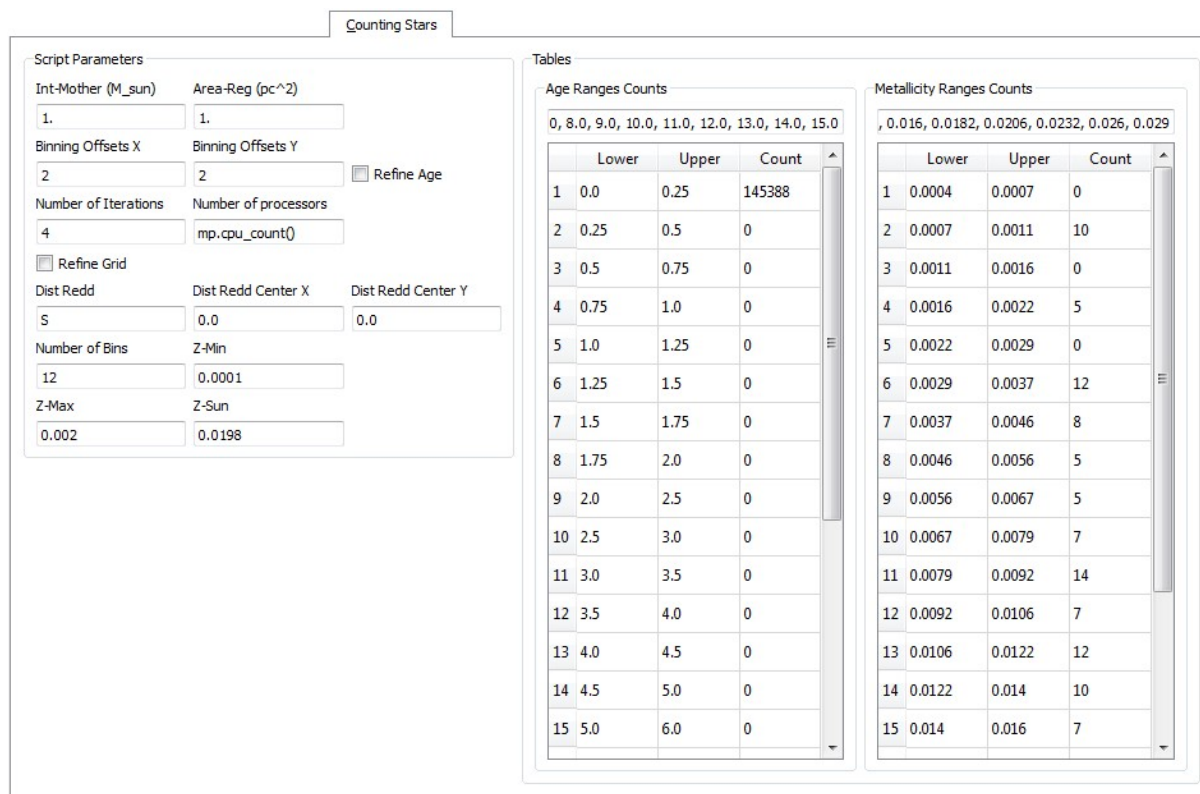


Figura 10.40: Presentación de la pestaña Counting Stars

- **Number of processors:** texto.
- **Refine Grid:** booleano determinado mediante una casilla de verificación.
- **Dist Redd:** texto.
- **Dist Redd Center X:** texto.
- **Dist Redd Center Y:** texto.

Los cuatro parámetros siguientes se utilizan para calcular el número de estrellas que hay en los rangos de metalicidad:

- **Number of Bins:** se trata de un valor numérico de tipo entero.
- **Z-Min:** se trata de un valor numérico de tipo real.
- **Z-Max:** se trata de un valor numérico de tipo real.
- **Z-Sun:** se trata de un valor numérico de tipo real.

Finalmente hay dos líneas de texto donde se introducirán valores numéricos de tipo real separados por comas. Estos valores tomados de dos en dos (repetiendo el último valor tomado) constituyen los distintos rangos que se mostrarán en las tablas indicando el número de estrellas:

- **Age:** línea de texto para los rangos de edad, dentro de **Age Ranges Counts**.
- **Metallicity:** línea de texto para los rangos de metalicidad, dentro de **Metallicity Ranges Counts**.

Los valores que se introduzcan en estas líneas de texto deben estar ordenados de menor a mayor de izquierda a derecha.

Todos estos parámetros se guardan en los ficheros *.in* y *.ini* al seleccionar las acciones que generan estos ficheros.

Para acceder rápidamente a esta pestaña se puede usar el atajo **Alt+C**.

Para profundizar en el uso de esta pestaña se puede consultar la sección 10.5.3.14.

10.5.2.2.5 Ventanas flotantes Existen tres ventanas flotantes que el usuario puede reubicar, agrupar, mostrar, ocultar y separarla en una ventana independiente según desee. Sin embargo, es necesario advertir al usuario que agrupar las *ventanas flotantes* en pestañas traerá comportamientos no deseados.

Estas *ventanas flotantes* guardan una relación directa con la pestaña Showing Diagrams de la *ventana central* y las acciones Show/Hide diagram tool, Show/Hide bundle tool y Show/Hide histogram tool.

A continuación se describen por separado las tres ventanas flotantes:

Diagram tool En esta ventana, ilustrada en la figura 10.41, se concentran las funcionalidades que permiten alterar el aspecto de los diagramas color-magnitud de la pestaña *Showing Diagrams* de la ventana central. Estas funcionalidades son:

- **ColorBar**: es una caja seleccionable que muestra o no la barra de colores de los diagramas.
- **Scale**: es una lista desplegable que permite elegir la escala de los datos representados en los diagramas. Los posibles valores son **linear** (lineal), **log** (logarítmica), **sqrt** (raíz cuadrada).
- **Minimum range**: establece el valores mínimos que tendrán los ejes de los diagramas. Es importante que el usuario rellene este campo con valores numéricos válidos.
- **Maximum range**: establece el valores máximos que tendrán los ejes de los diagramas. Es importante que el usuario rellene este campo con valores numéricos válidos.
- **Bin size**: establece los valores que tendrá el tamaño de la rejilla de los diagramas.

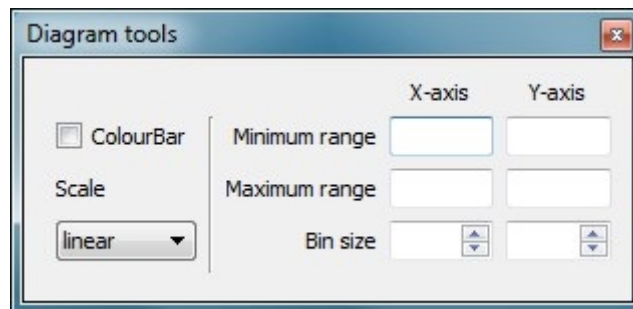


Figura 10.41: Presentación de la ventana flotante Diagram tool

Bundle tool Esta ventana, figura 10.42, permite al usuario realizar distintas operaciones sobre bundles.

En esta ventana se dispone de los siguientes elementos:

- Una *lista desplegable* que expandiéndola mostrará todos bundles definidos. Se podrá elegir uno de ellos como bundle seleccionado.
- Una *tabla* que mostrará los puntos que contiene el bundle seleccionado.
- Dos *cuadros numéricos* donde se mostrará el tamaño de la rejilla del bundle activo.
- Dos *cuadros numéricos* donde se mostrarán las coordenadas de los puntos a añadir o modificar.
- *Botones* relacionados con las acciones.

Las acciones que se pueden realizar sobre bundles, además de mostrar su información, son:

- Crear bundle: definida en *Create bundle*.
- Cerrar bundle: definida en *Close bundle*.

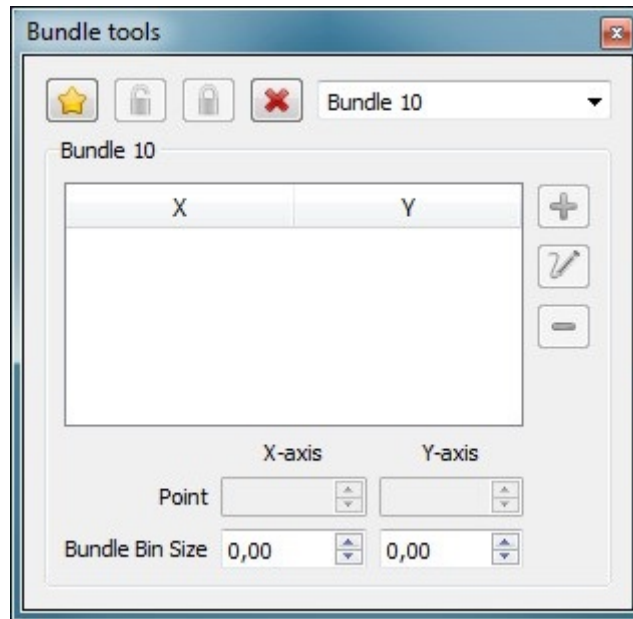


Figura 10.42: Presentación de la ventana flotante Bundle tool

- Abrir bundle: definida en *Open bundle*.
- Eliminar el bundle completo: definida en *Remove bundle*.
- Modificar los puntos del bundle: definida en *Modify point*.
- Eliminar un punto del bundle: definida en *Remove point*.
- Modificar el tamaño de la rejilla del bundle: usando los controles de los cuadros numéricos (Bundle Bin Size) o mediante teclado se puede establecer el tamaño de la rejilla del bundle.
- Añadir puntos a los bundles: hay varias formas de añadir puntos al bundle seleccionado. Dada la importancia de esta acción se recomienda consultar la sección 10.5.3.3.

En ambos diagramas serán resaltados los puntos del bundle seleccionado con un grosor de punto mayor en color violeta.

En la lista desplegable se resaltará cada bundle definido con un color de fondo de la siguiente forma:

- Color verde: el bundle está *cerrado*.
- Color amarillo: el bundle está *abierto y listo para ser cerrado* porque contiene tres o más puntos.
- Color rojo: el bundle está *abierto y no está listo para ser cerrado* porque contiene dos o menos puntos.

Histogram tool Esta ventana flotante, ilustrada en la figura 10.43, permite al usuario visualizar dos histogramas de los bundles que cumplen ciertas condiciones.

Las condiciones que deben cumplir los bundles para que sean representados en los histogramas son que su estado actual sea el de cerrado y los valores de su rejilla sean mayores al cero absoluto.

Si no existe ningún bundle que cumpla estas condiciones, se deshabilitará la acción *Histogram tool*.

Habrán dos histogramas, uno por cada diagrama color-magnitud de la pestaña *Showing Diagrams* de la ventana central.

Los histogramas se generan cuando el usuario pulse el botón **Draw histograms**.

Existe una leyenda que situada en la esquina superior derecha de cada histograma que permite identificar cada bundle representado. Se asignó un nombre corto a cada bundle que sigue la notación Bdd, donde dd es la numeración del bundle.

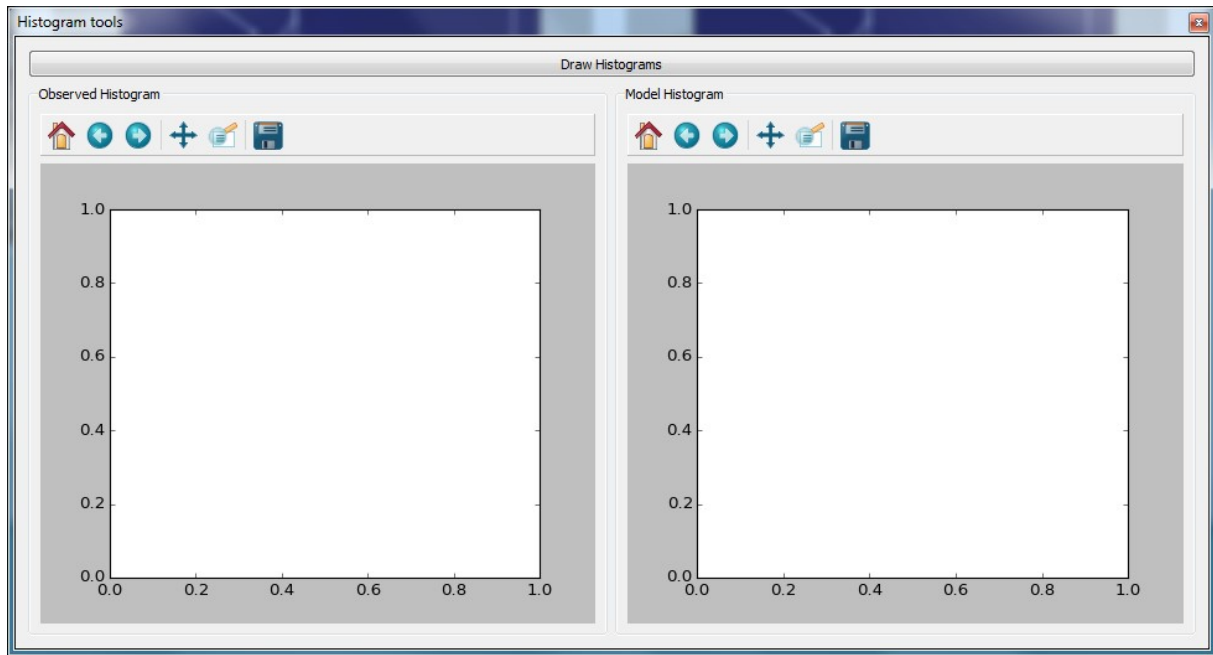


Figura 10.43: Presentación de la ventana flotante Histogram tool

El usuario puede hacer clic sobre cada bundle en la leyenda para mostrarlo u ocultarlo en el histograma, según desee.

10.5.3 Casos de uso

En esta sección se enseñará al usuario a utilizar las funcionalidades más importantes de la aplicación utilizando capturas de pantalla.

10.5.3.1 Carga de datos

La figura 10.44 muestra el proceso que debe seguir el usuario para cargar los datos en la aplicación:

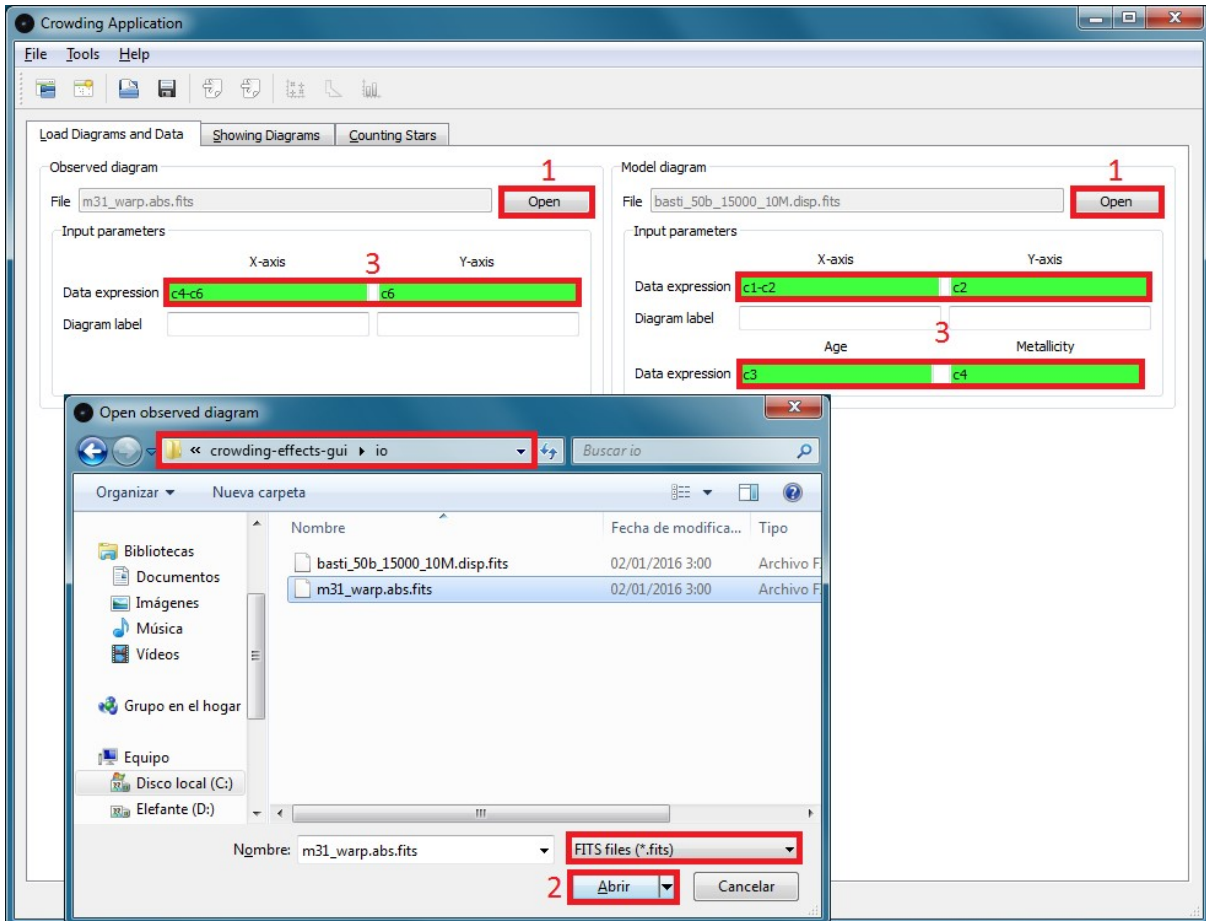


Figura 10.44: Ejemplo de uso de Carga de datos.

El proceso que debe seguir el usuario consta de los siguientes pasos:

1. Hacer clic en el botón **Open** para desplegar el cuadro de diálogo con el que seleccionaremos el fichero.
2. Navegamos por el árbol de directorios del sistema, filtrando o no los archivos por extensión *FITS*, hasta localizar el fichero deseado, lo seleccionamos y pulsamos **Abrir**.
3. Finalmente rellenamos las líneas de texto de las expresiones. No es necesario seguir ningún orden al rellenar las líneas de texto para cargar las expresiones.

10.5.3.2 Creación de un bundle

La figura 10.45 muestra el proceso que debe seguir el usuario para crear un bundle después de haber cargado los datos.

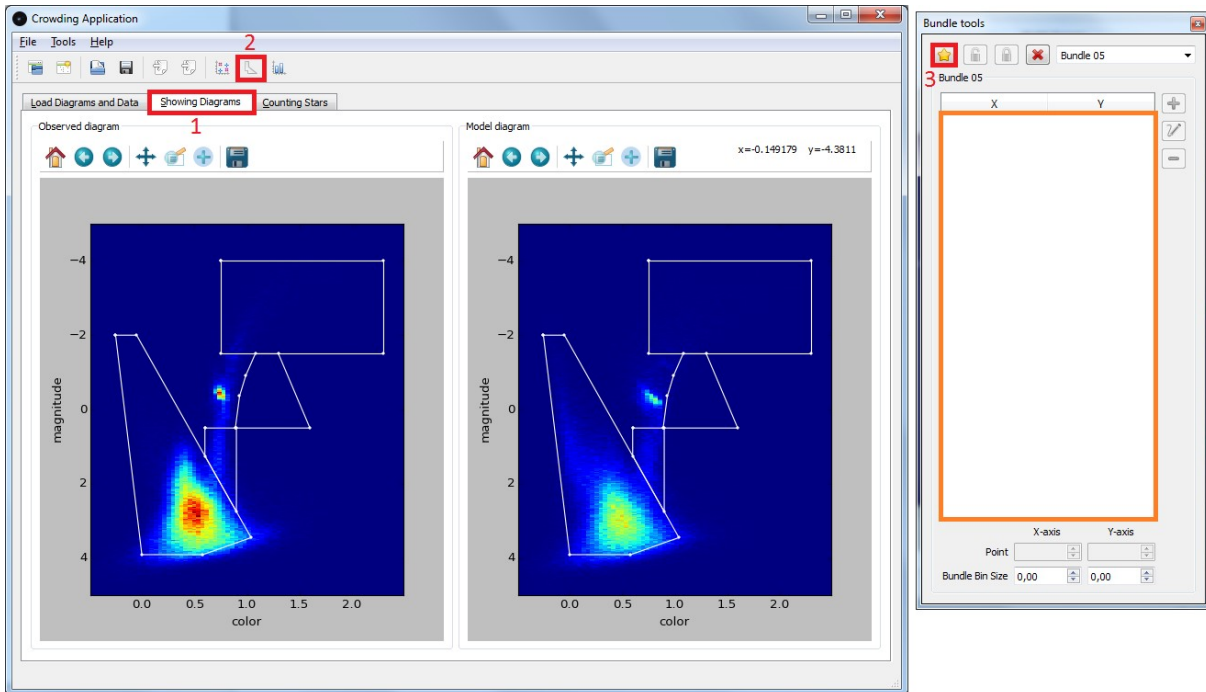


Figura 10.45: Ejemplo de uso de Creación de un bundle.

El proceso que debe seguir el usuario consta de los siguientes pasos:

1. Situar en la pestaña *Showing Diagrams*.
2. Seleccionar la acción *Show/Hide bundle tool* para mostrar la ventana flotante *Bundle tool*.
3. En la ventana flotante seleccionamos la acción *Create bundle*, de esta forma se ha creado un nuevo bundle sin puntos como se puede ver en el recuadro **naranja** de la figura 10.45.

10.5.3.3 Añadir punto a un bundle

En esta sección se mostrarán los distintos procedimientos que permiten al usuario añadir puntos a un bundle.

Antes de insertar un punto se mostrará con un ejemplo los pasos preliminares:

- Seleccionar el bundle al que se desea añadir un punto. Hay que **seleccionar un *bundle* abierto**. Los *bundles* abiertos (1) se muestran en la figura 10.46.

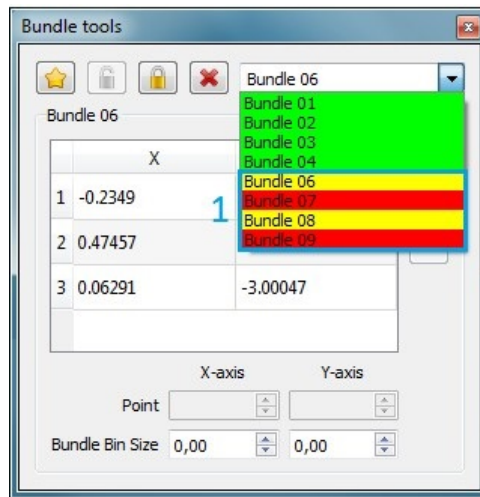


Figura 10.46: Ejemplo de uso de *Añadir punto a un bundle*, seleccionar bundle abierto.

- Al seleccionar el Bundle 06, los puntos del bundle seleccionado (6) se resaltarán en los diagramas color-magnitud como podemos ver en la figura 10.47.

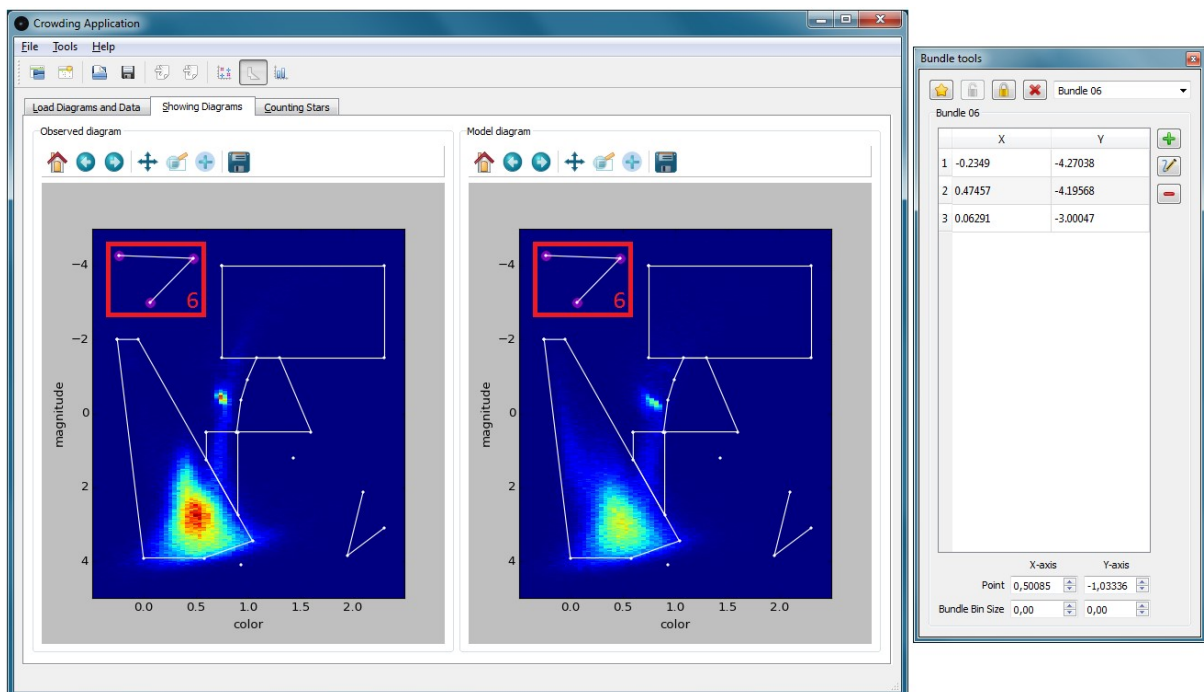


Figura 10.47: Ejemplo de uso de *Añadir punto a un bundle*, bundle seleccionado.

Sobre el bundle seleccionado anteriormente (Bundle 06) se añadirán cuatro puntos para mostrar las distintas formas disponibles para añadir puntos:

10.5.3.3.1 Botón izquierdo del ratón

1. Situamos el ratón dentro de uno de los diagramas color-magnitud en la coordenada deseada. Como podemos ver en la figura 10.48 el puntero del ratón se convierte en una cruz (1) y la coordenada en la que nos encontramos se muestra (2).

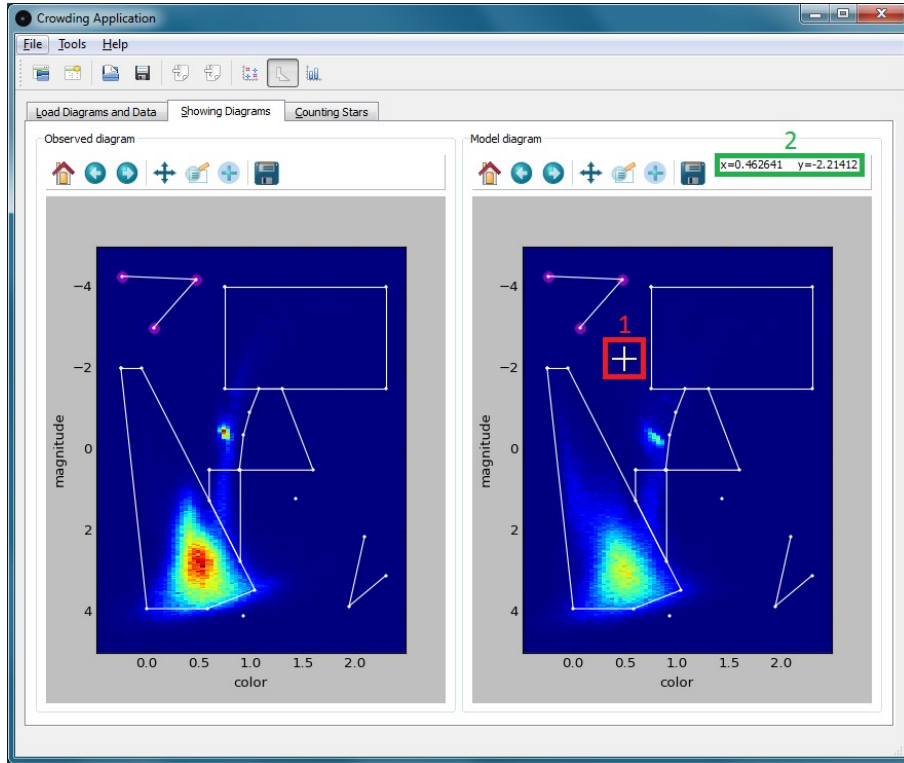


Figura 10.48: Ejemplo de uso de *Añadir punto a un bundle*, puntero.

2. Hacemos clic con el botón izquierdo del ratón. Como se puede ver en la figura 10.49 se ha añadido un punto en cada diagrama (1) y la tabla (2) muestra las coordenadas del punto.

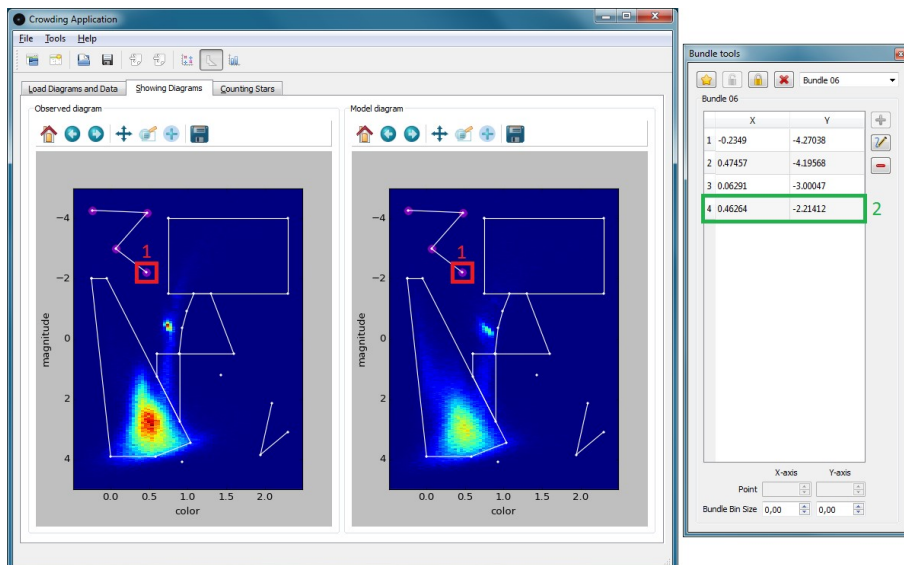


Figura 10.49: Ejemplo de uso de *Añadir punto a un bundle*, caso 10.5.3.3.1.

10.5.3.3.2 Botón derecho del ratón

1. Desplazar el ratón sobre uno de los diagramas color-magnitud. Al llegar al punto deseado hacer clic con el botón derecho. La figura 10.50 ilustra las coordenadas del punto (1), mostradas en (2) se cargan en (3) para su edición. Cuando se finalice la edición se pulsa en (4) para añadir el punto.

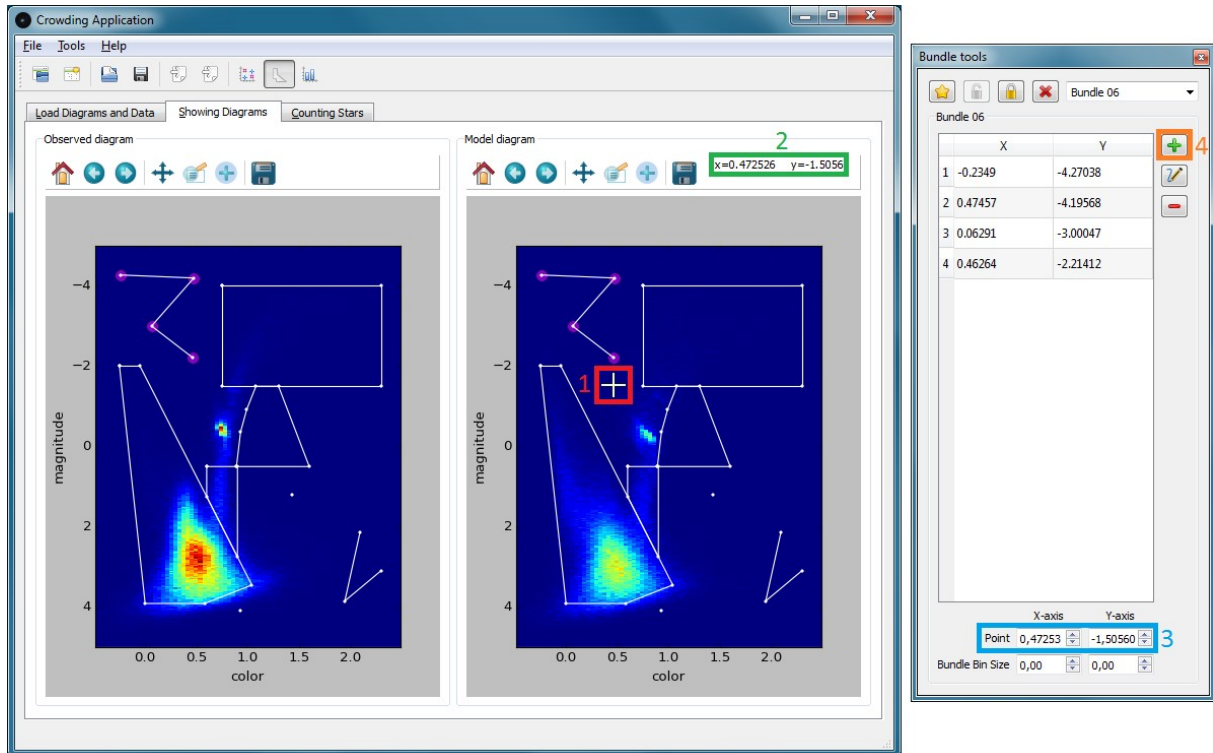


Figura 10.50: Ejemplo de uso de *Añadir punto a un bundle*, caso 10.5.3.3.2.

10.5.3.3.3 Botón izquierdo del ratón sobre una recta El punto que se quiere añadir pertenece a cualquier línea que une dos puntos de otro bundle.

1. Para hacer uso de esta funcionalidad hay que seleccionar la acción *Select a point* (1). Al desplazar el ratón sobre el diagrama, *para el que se activó la acción*, se puede ver que el puntero del ratón tiene forma de mano (2). Las coordenadas del punto y la acción seleccionada se muestran en (3). Ilustrado en la figura 10.51.

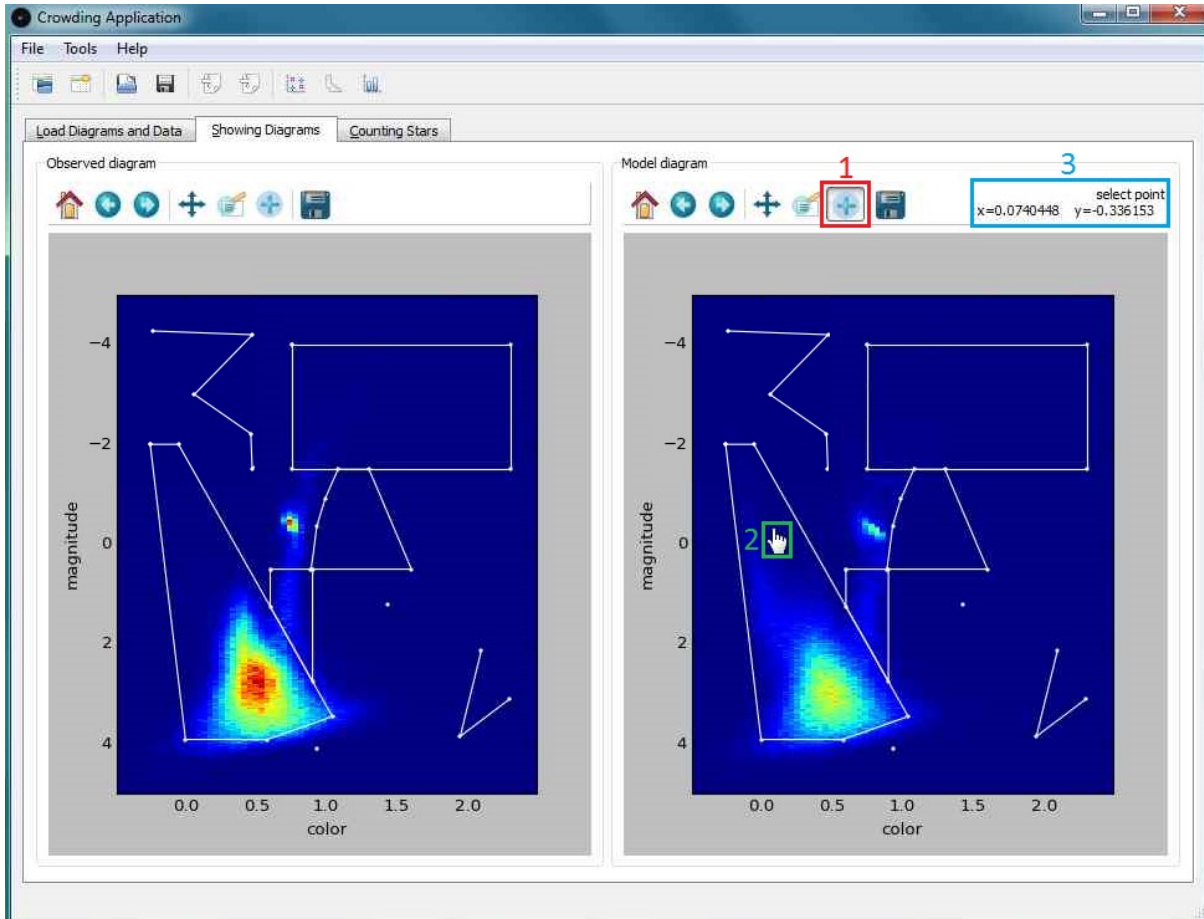


Figura 10.51: Ejemplo de uso de *Añadir punto a un bundle*, *Select a point*

2. Situar el puntero del ratón sobre alguna de las líneas blancas del diagrama. Por ejemplo la línea señalada por (1) en el punto señalado por (2). Las coordenadas del punto se muestran en (3). Ilustrado en la figura 10.52.
3. Después de hacer clic con el botón izquierdo del ratón se añadirá el punto a ambos diagramas (1) y las coordenadas del punto se muestran en (2). El puntero del ratón vuelve a ser una cruz (3) y la acción *Select a point* se desactiva como se puede ver en (4). Ilustrado en la figura 10.53.

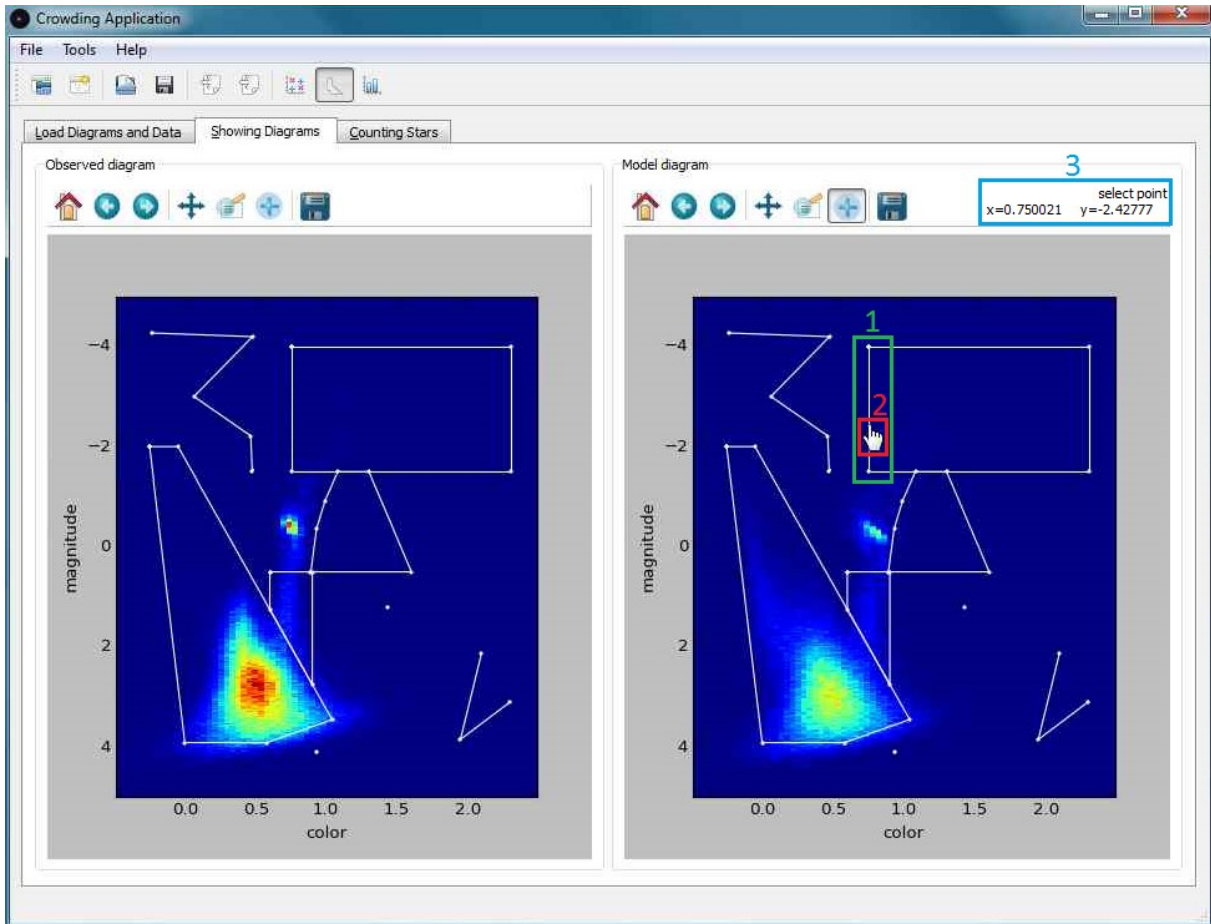


Figura 10.52: Ejemplo de uso de *Añadir punto a un bundle*, caso 2.

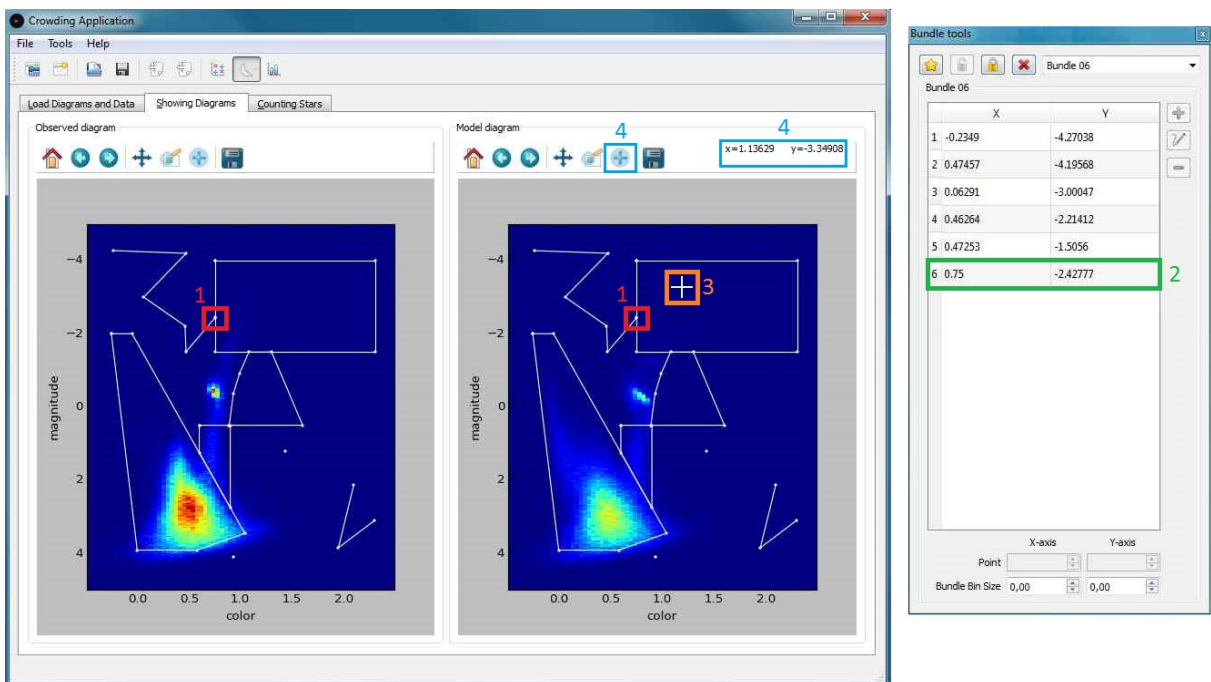


Figura 10.53: Ejemplo de uso de *Añadir punto a un bundle*, caso 3.

10.5.3.3.4 Botón izquierdo del ratón sobre un punto existente

1. Se procede como en la descripción del punto 1 del apartado 10.5.3.3.3, activando la acción *Select a point*.
2. Situar el puntero del ratón sobre alguno de los puntos ya existentes. Por ejemplo el punto señalado por el puntero del ratón en (1). Las coordenadas del punto se muestran en (2). Ilustrado en la figura 10.54

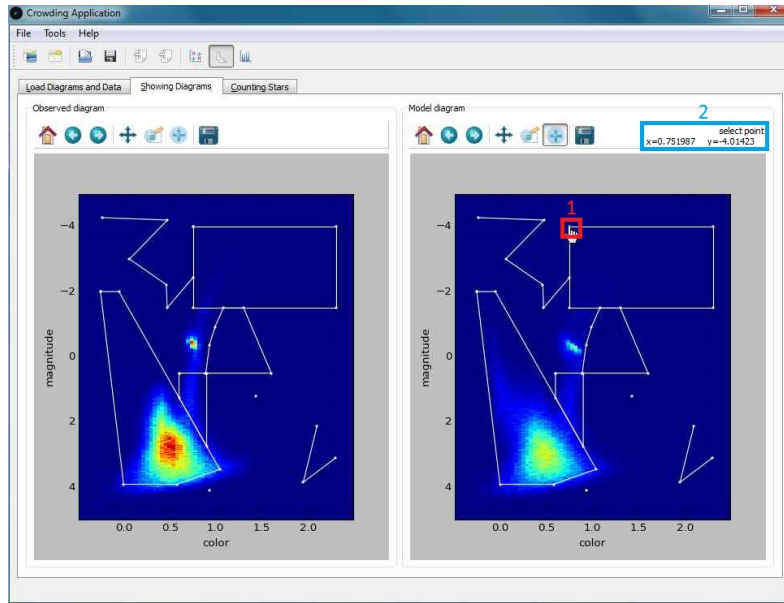


Figura 10.54: Ejemplo de uso de *Añadir punto a un bundle*, caso 2.

3. Después de hacer clic con el botón izquierdo del ratón el punto se añadirá al bundle seleccionado como se puede ver en (1). El puntero del ratón vuelve a ser una cruz (2) y la acción *Select a point* se desactiva como se puede ver en (3). Ilustrado en la figura 10.55.

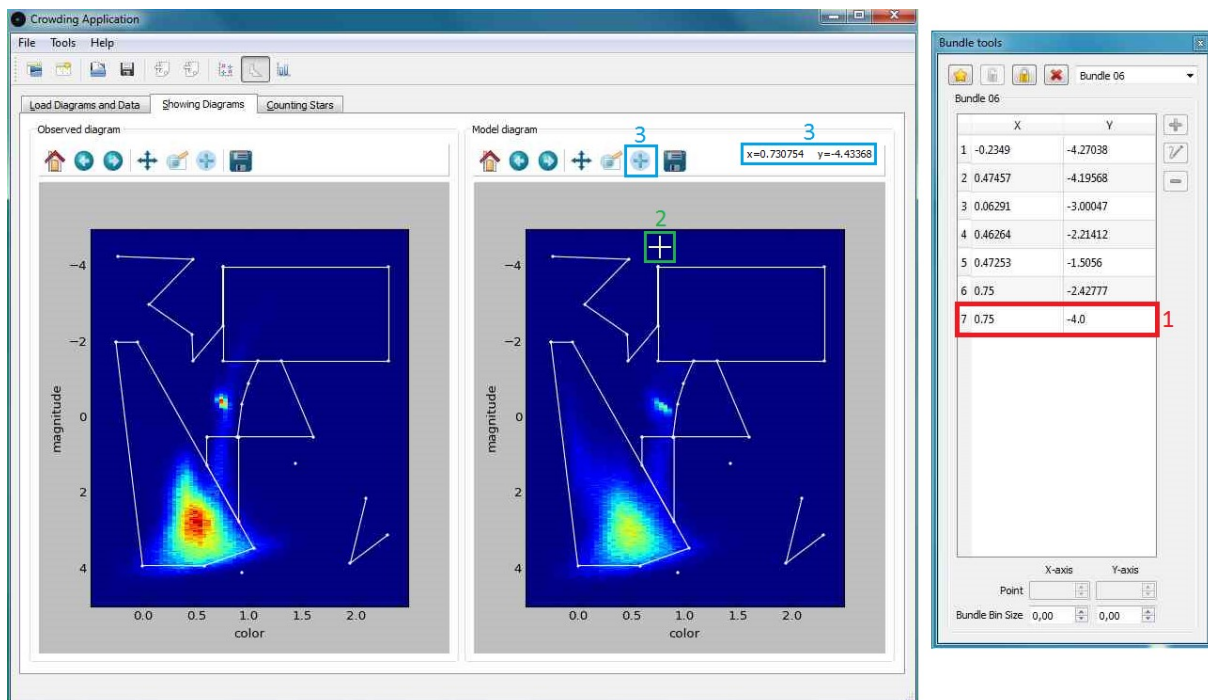


Figura 10.55: Ejemplo de uso de *Añadir punto a un bundle*, caso 3.

10.5.3.4 Modificar un punto de un bundle

La figura 10.56 muestra el proceso que debe seguir el usuario para modificar un punto de un bundle:

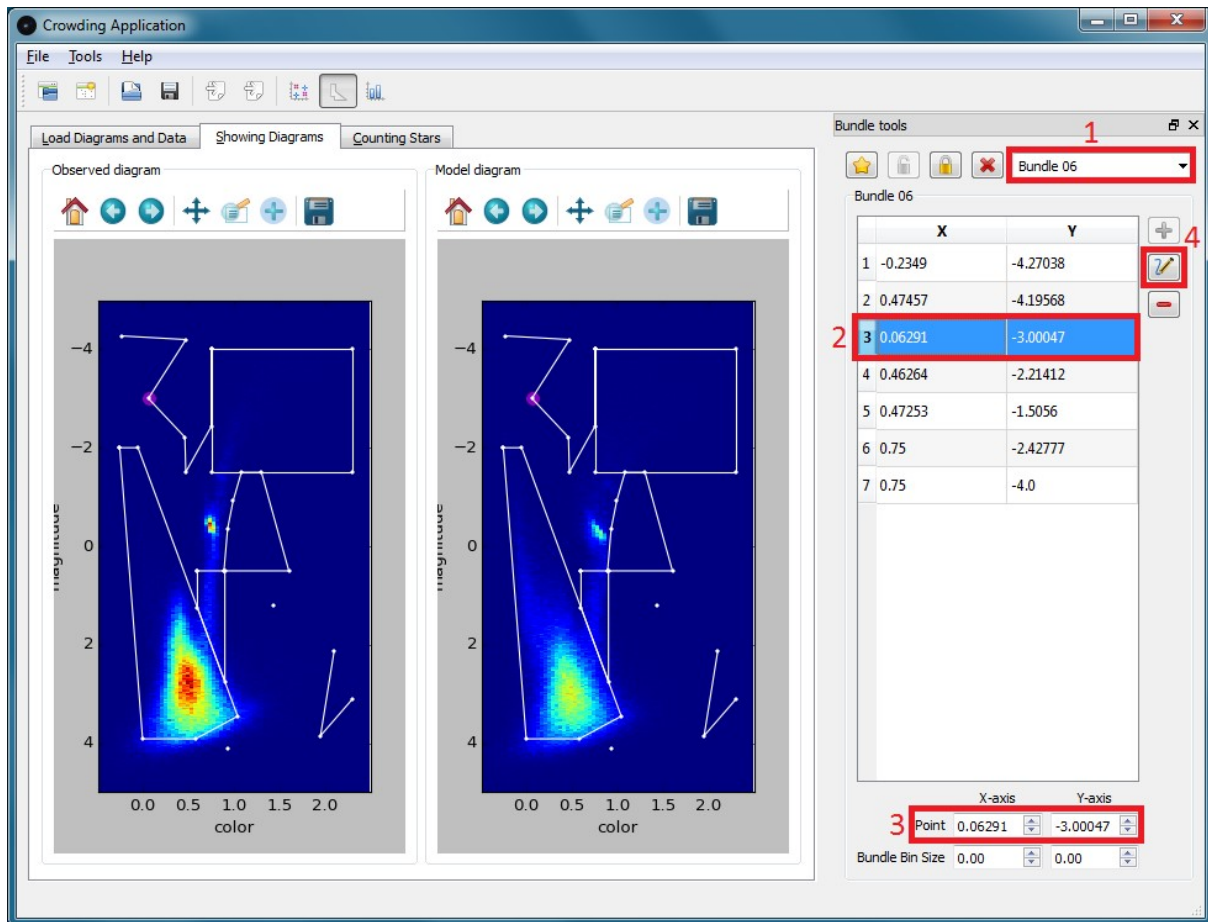


Figura 10.56: Ejemplo de uso de *Modificar un punto de un bundle*.

Los pasos que debe seguir el usuario para modificar un punto de un bundle son:

1. Seleccionar el bundle en el que se modificará el punto, usando para ello la lista desplegable.
2. Se escoge el punto que se quiere modificar haciendo clic con el botón izquierdo del ratón dentro de la tabla.
3. Las coordenadas del punto escogido se cargan en las líneas numéricas. Se podrán editar las coordenadas del punto mediante los controles de las líneas o con el teclado.
4. Una vez que se haya terminado con la edición se confirma la modificación haciendo clic sobre icono asociado con la acción *Modify point*.

10.5.3.5 Eliminar punto de un bundle

La figura 10.57 muestra el proceso que debe seguir el usuario para eliminar un punto de un bundle.

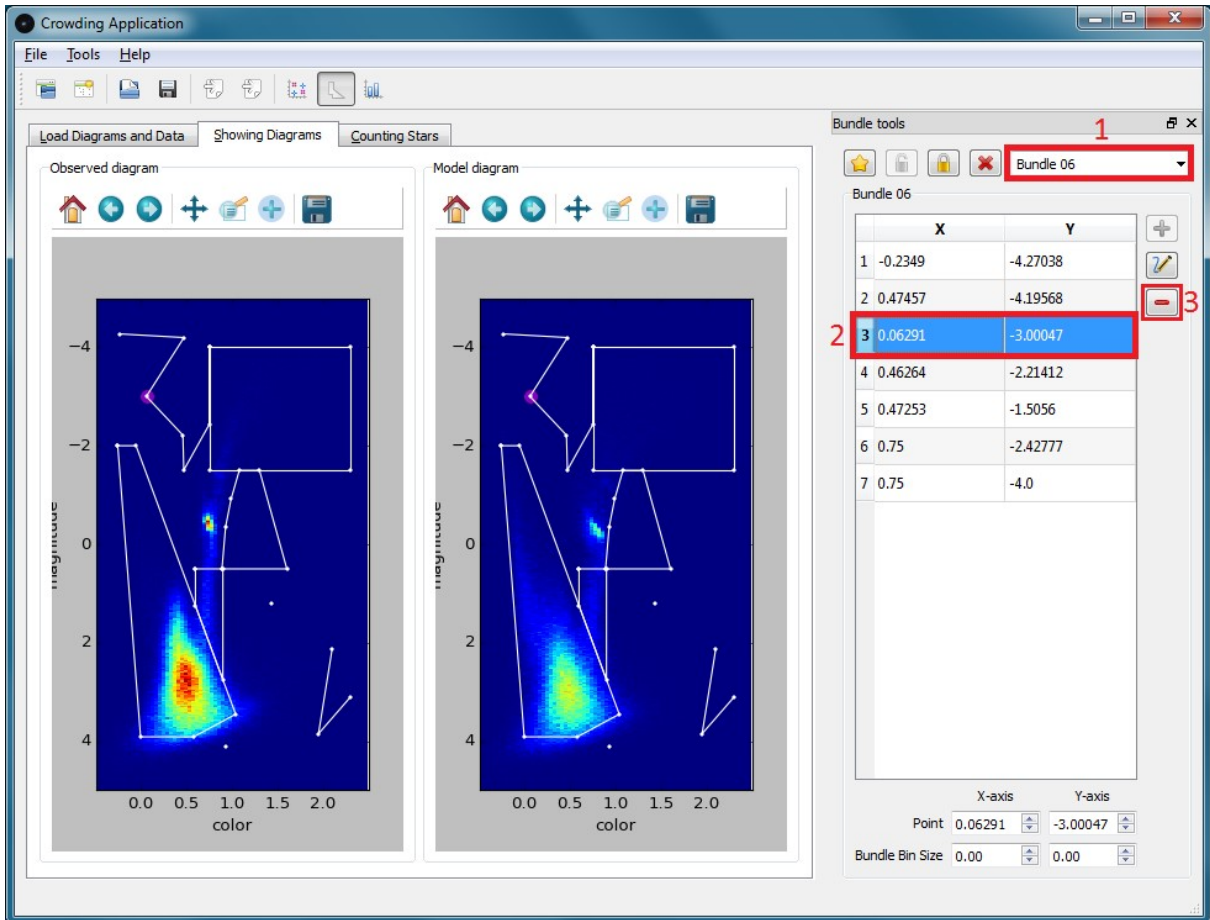


Figura 10.57: Ejemplo de uso de *Eliminar punto de un bundle*.

Los pasos que debe seguir el usuario para eliminar un punto de un bundle son:

1. Seleccionar el bundle del que vamos a eliminar el punto, usando para ello la lista desplegable.
2. Se escoge el punto que se quiere eliminar haciendo clic con el botón izquierdo del ratón dentro de la tabla.
3. Se hace clic con el botón izquierdo del ratón sobre el icono asociado con la acción *Remove point*.

10.5.3.6 Abrir un bundle

La figura 10.58 muestra el proceso que debe seguir el usuario para abrir un bundle.

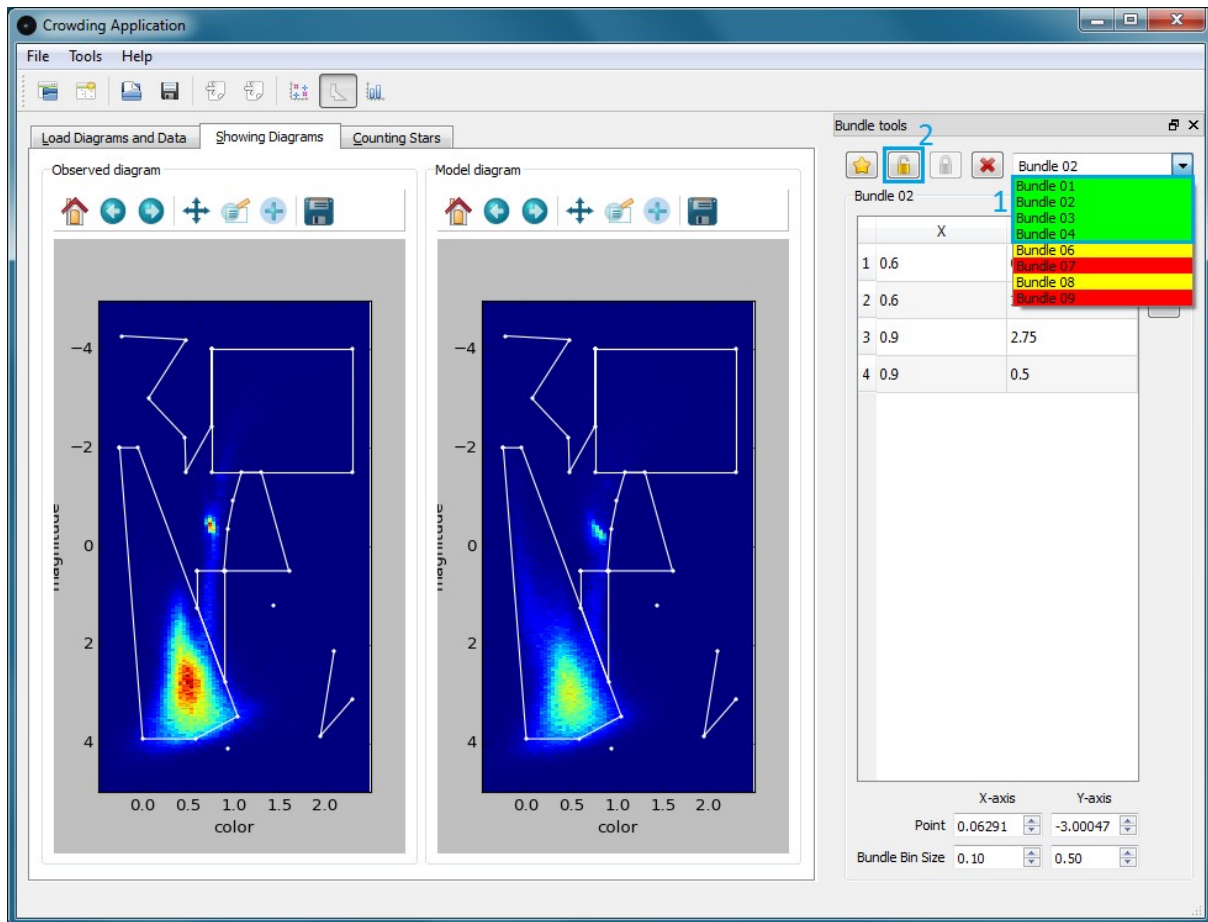


Figura 10.58: Ejemplo de uso de *Abrir un bundle*.

El proceso que debe seguir el usuario para abrir un bundle es el siguiente:

1. Se debe desplegar la lista de bundles y escoger un bundle que esté cerrado. Los bundles cerrados se muestran en *color verde* en la lista desplegable.
2. Una vez seleccionado el bundle se pulsa el icono asociado con la acción *Open bundle*.

10.5.3.7 Eliminar un bundle

La figura 10.59 muestra el proceso que debe seguir el usuario para eliminar un bundle.

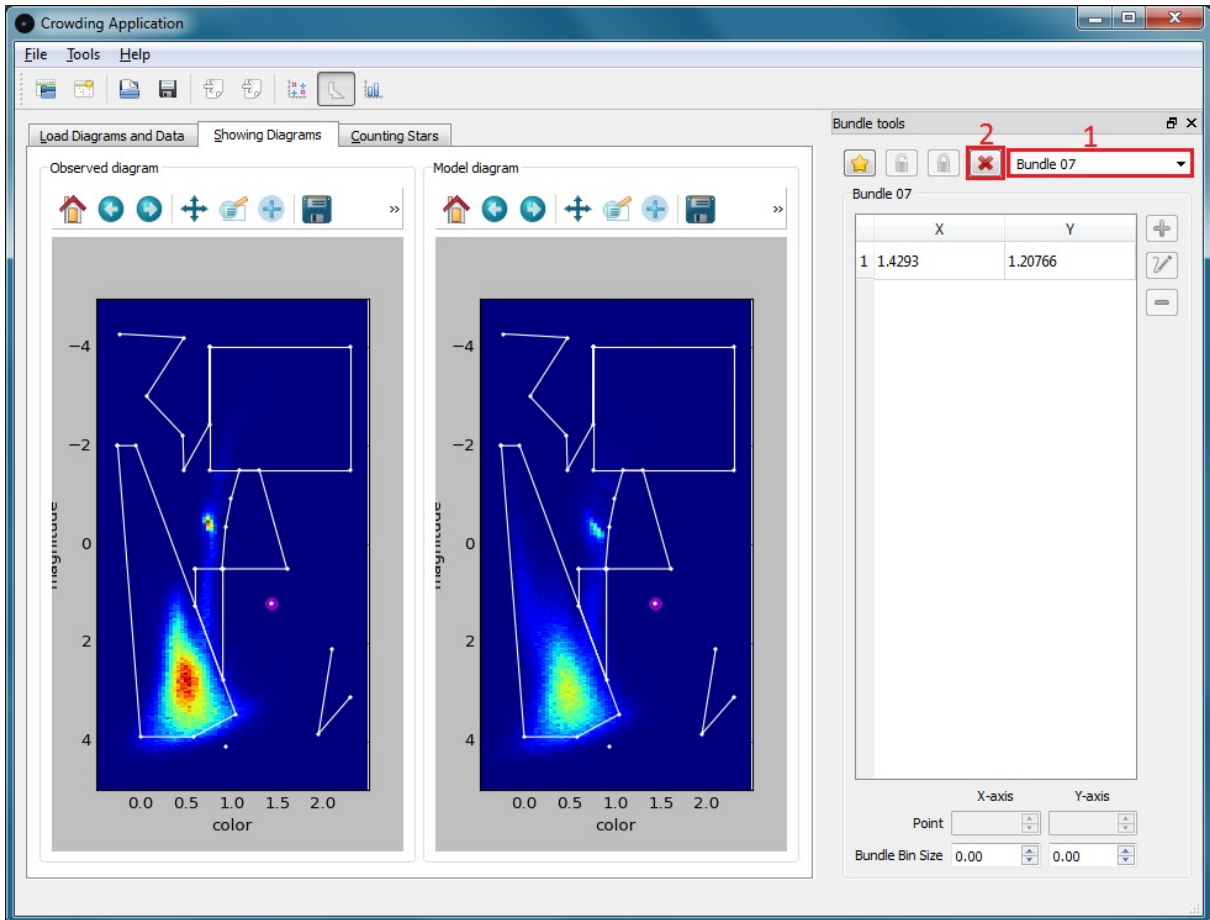


Figura 10.59: Ejemplo de uso de *Eliminar un bundle*.

El proceso que debe seguir el usuario para abrir un bundle es el siguiente:

1. Desplegar la lista de bundles y escoger el bundle que queremos eliminar.
2. Una vez seleccionado el bundle se pulsa el icono asociado con la acción *Remove bundle*.

10.5.3.8 Cerrar un bundle

La figura 10.60 muestra el proceso que debe seguir el usuario para cerrar un bundle.

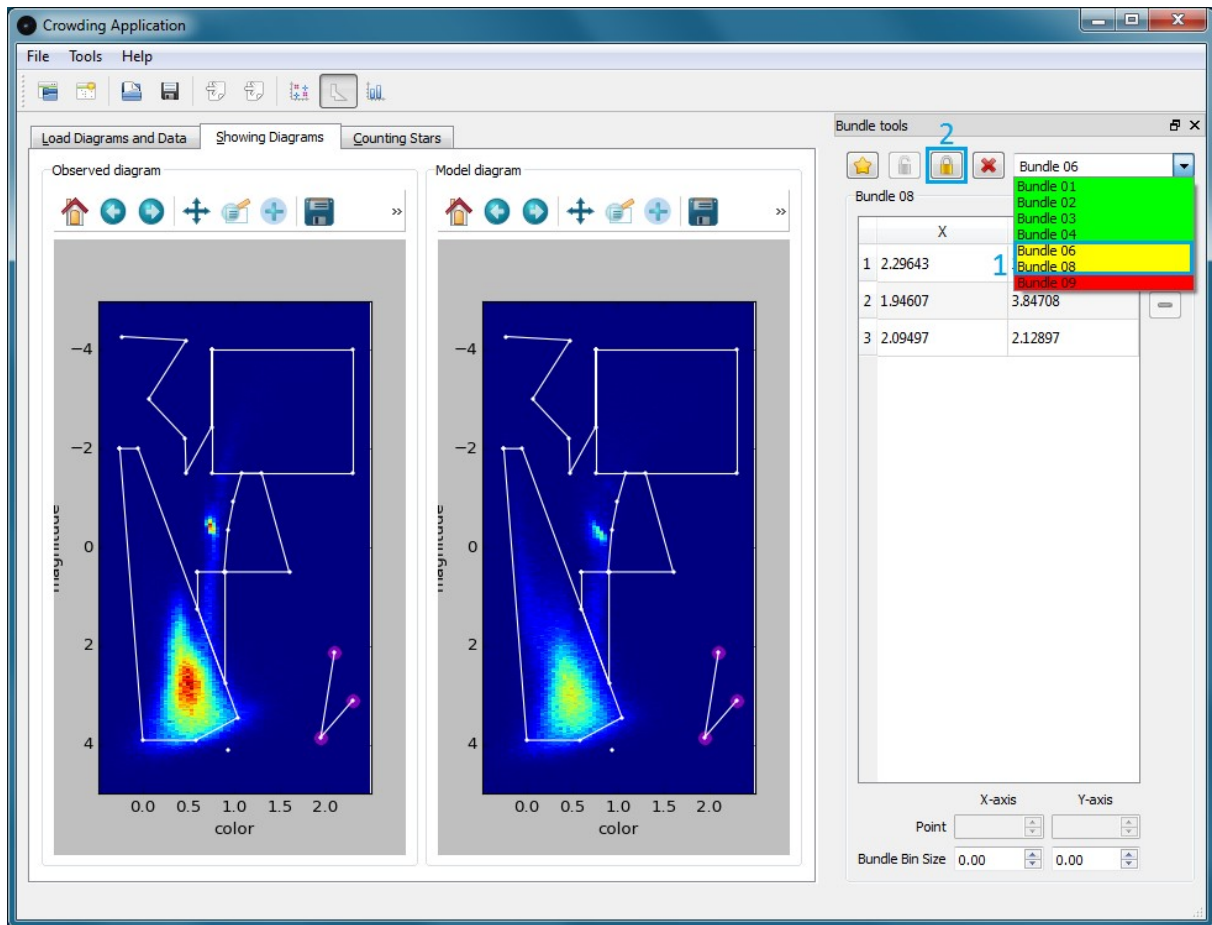


Figura 10.60: Ejemplo de uso de *Cerrar un bundle*.

El proceso que debe seguir el usuario para cerrar un bundle es el siguiente:

1. Se debe desplegar la lista de bundles y escoger un bundle que esté abierto y listo para ser cerrado. Los bundles que cumplen estas condiciones se muestran en color amarillo en la lista desplegable.
2. Una vez seleccionado el bundle se pulsa el icono asociado con la acción *Close bundle*.

10.5.3.9 Establecer la rejilla de un bundle

La figura 10.61 muestra el proceso que debe seguir el usuario para establecer el tamaño de *la rejilla de un bundle*: Bundle Bin Size X y Bundle Bin Size Y.

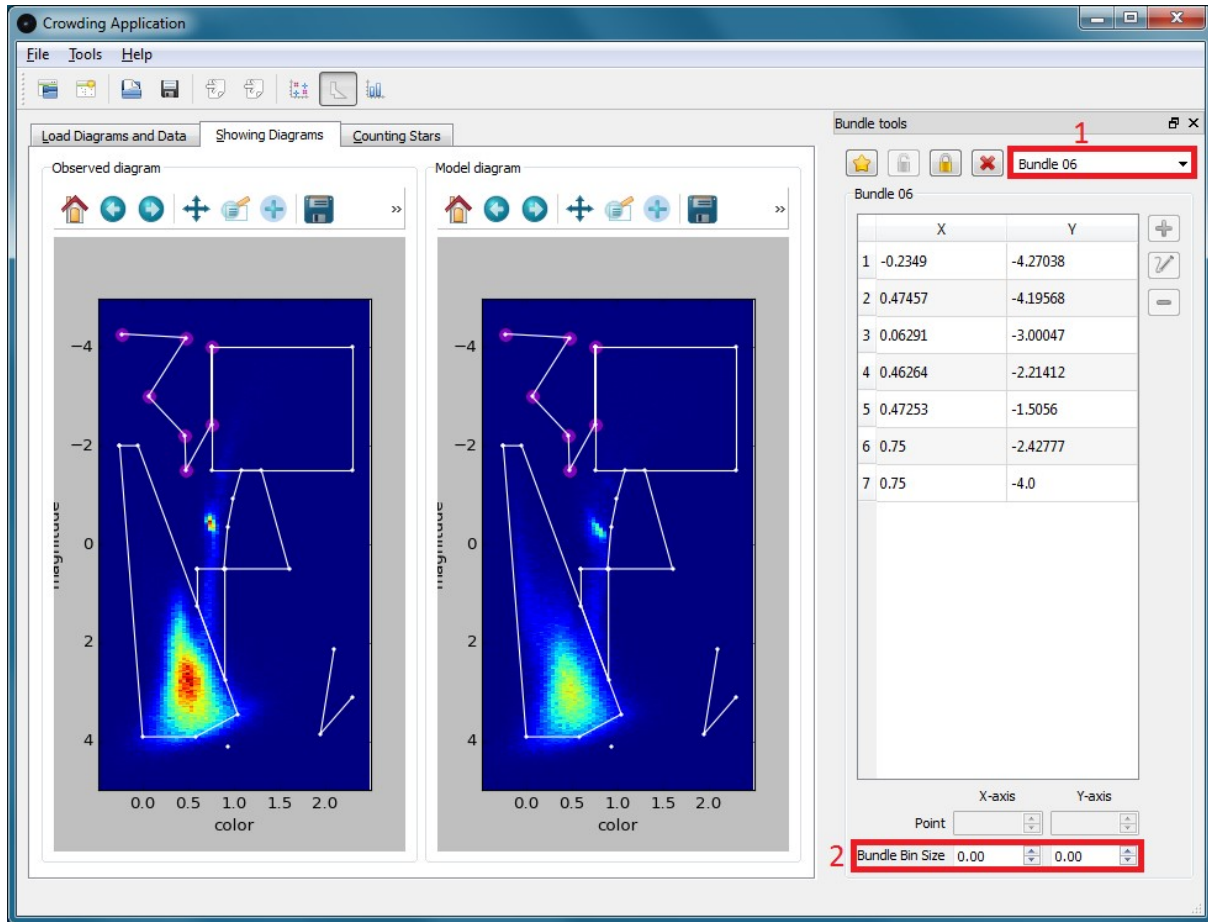


Figura 10.61: Ejemplo de uso de *Establecer la rejilla de un bundle*.

Los pasos que debe seguir el usuario para *establecer la rejilla de un bundle* son:

1. Seleccionar el bundle para el que se quiere establecer o modificar la rejilla. Por defecto al crear un bundle el tamaño de su rejilla (Bundle Bin Size X y Bundle Bin Size Y) se establece en cero absoluto.
2. Se usarán los cuadros numéricos para establecer el tamaño de la rejilla del bundle seleccionado. Para ello se pueden usar los controles de cada cuadro numérico o el teclado.

10.5.3.10 Modificar el aspecto de los diagramas

La figura 10.62 muestra el proceso que se debe seguir para desplegar la ventana que permite modificar el aspecto de los diagramas color-magnitud.

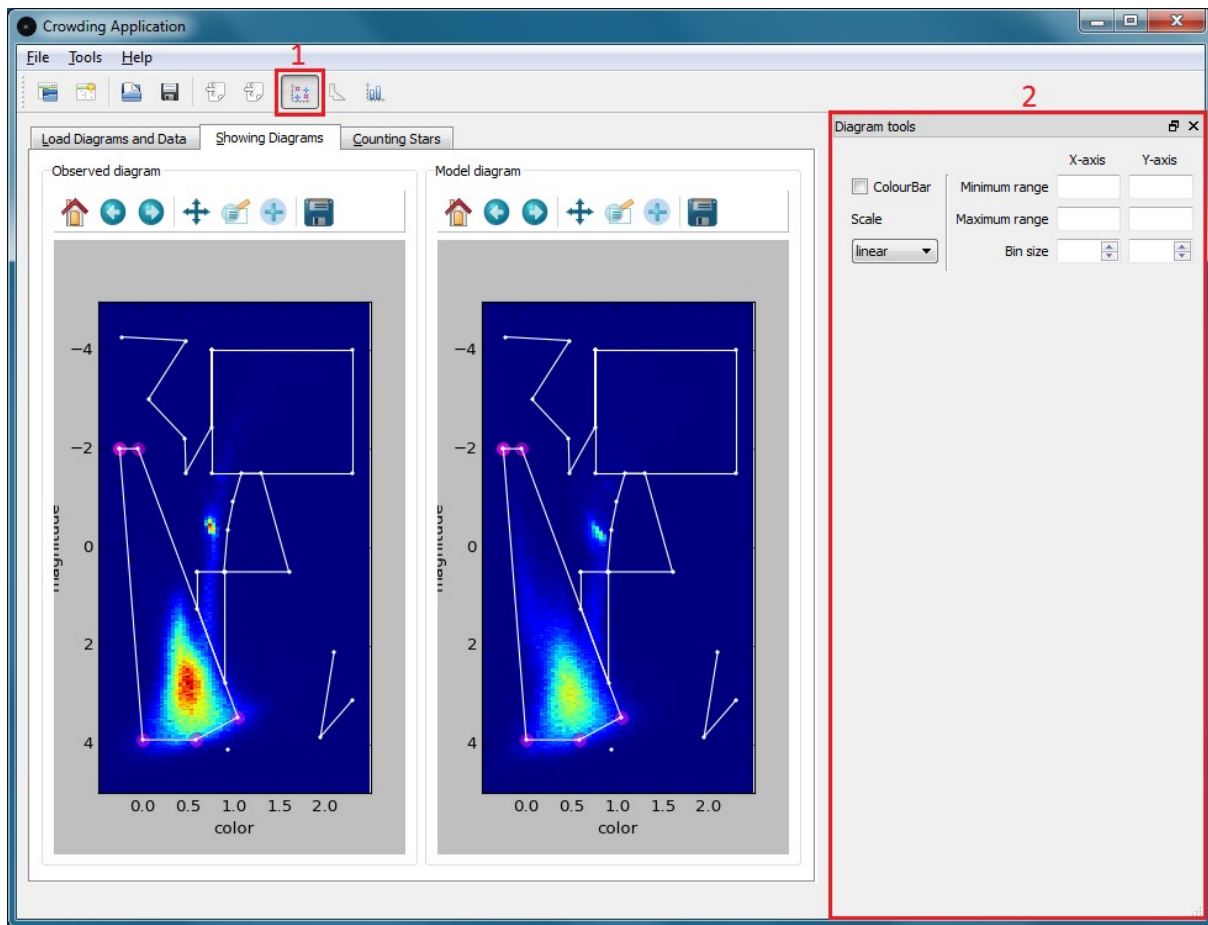


Figura 10.62: Ejemplo de uso de *Modificar el aspecto de los diagramas*.

Los pasos a seguir son:

1. Seleccionar el icono asociado con la acción *Show/Hide diagram tool* para mostrar la ventana flotante *Diagram tool*.
2. Se mostrará la ventana flotante *Diagram tool* con los elementos que modifican el aspecto de los diagramas color magnitud. El efecto de cada uno de los elementos sobre los diagramas se explican en la sección 10.5.2.2.5 en *Diagram tool*.

10.5.3.11 Mostrar histogramas de los bundles

La figura 10.63 muestra el proceso que debe seguir el usuario para ver los histogramas de los bundles cerrados con rejilla establecida:

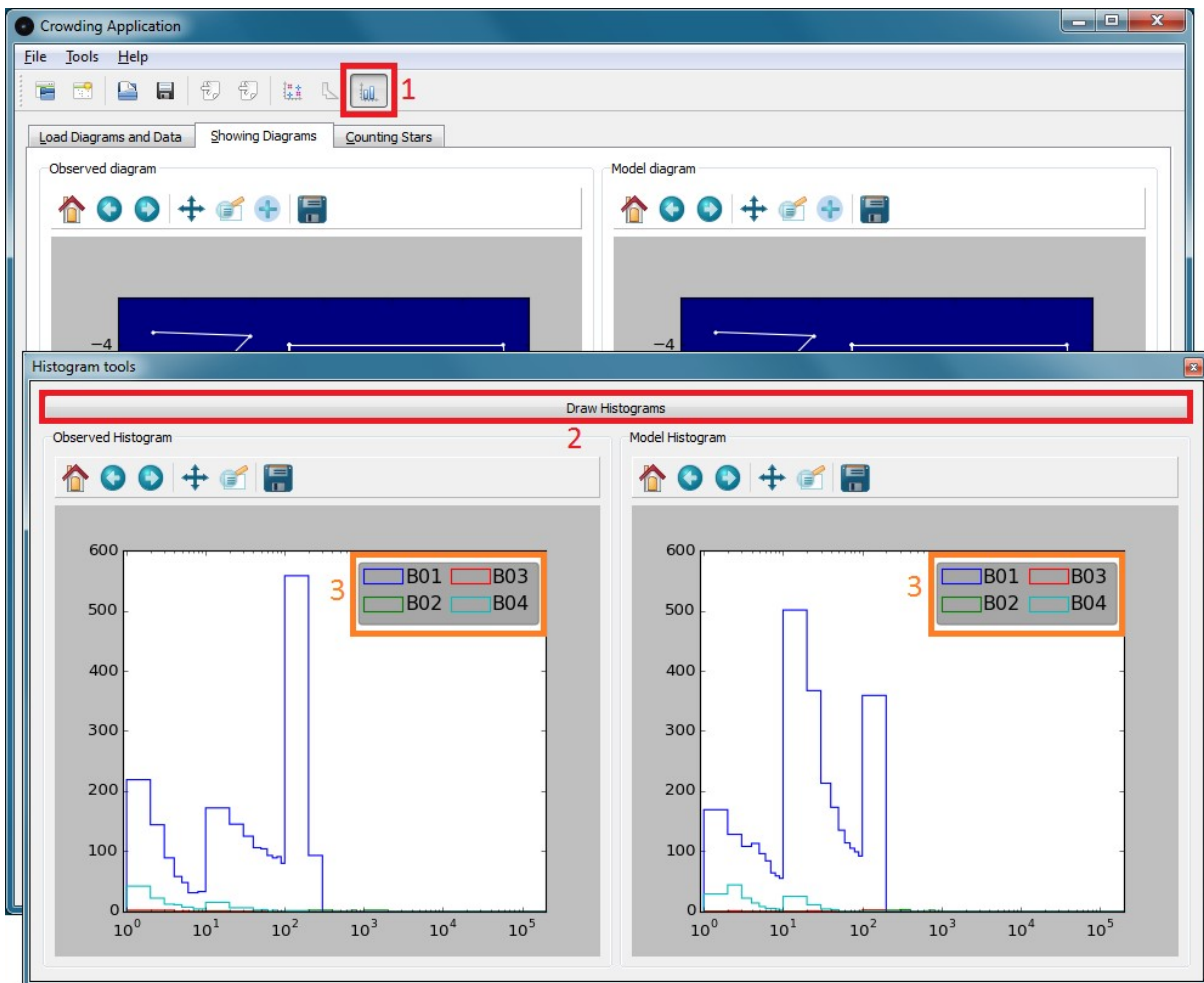


Figura 10.63: Ejemplo de uso de *Mostrar histogramas de los bundles*.

Los pasos a seguir son:

1. Seleccionar el icono asociado con la acción *Show/Hide histogram tool* para mostrar la ventana flotante *Histogram tool*. La ventana flotante se podrá desplegar *si hay al menos un bundle cerrado con rejilla establecida*.
2. Hacer clic en el botón *Draw Histograms*. Se mostrarán los histogramas de *los bundles cerrados con rejilla establecida*. Hay un histograma para los bundles del diagrama observado y otro para los del diagrama modelo.
3. Hay una leyenda que permite *mostrar/ocultar el histograma de un bundle*. Para ello hay que hacer clic sobre los cuadrados de color que hay junto al nombre de los bundles en la leyenda.

10.5.3.12 Salvar el estado de la aplicación

La figura 10.64 muestra el proceso que debe seguir el usuario para salvar el estado de la aplicación. Se puede salvar el estado de la aplicación de dos formas, mediante ficheros `.in` o ficheros `.ini`.

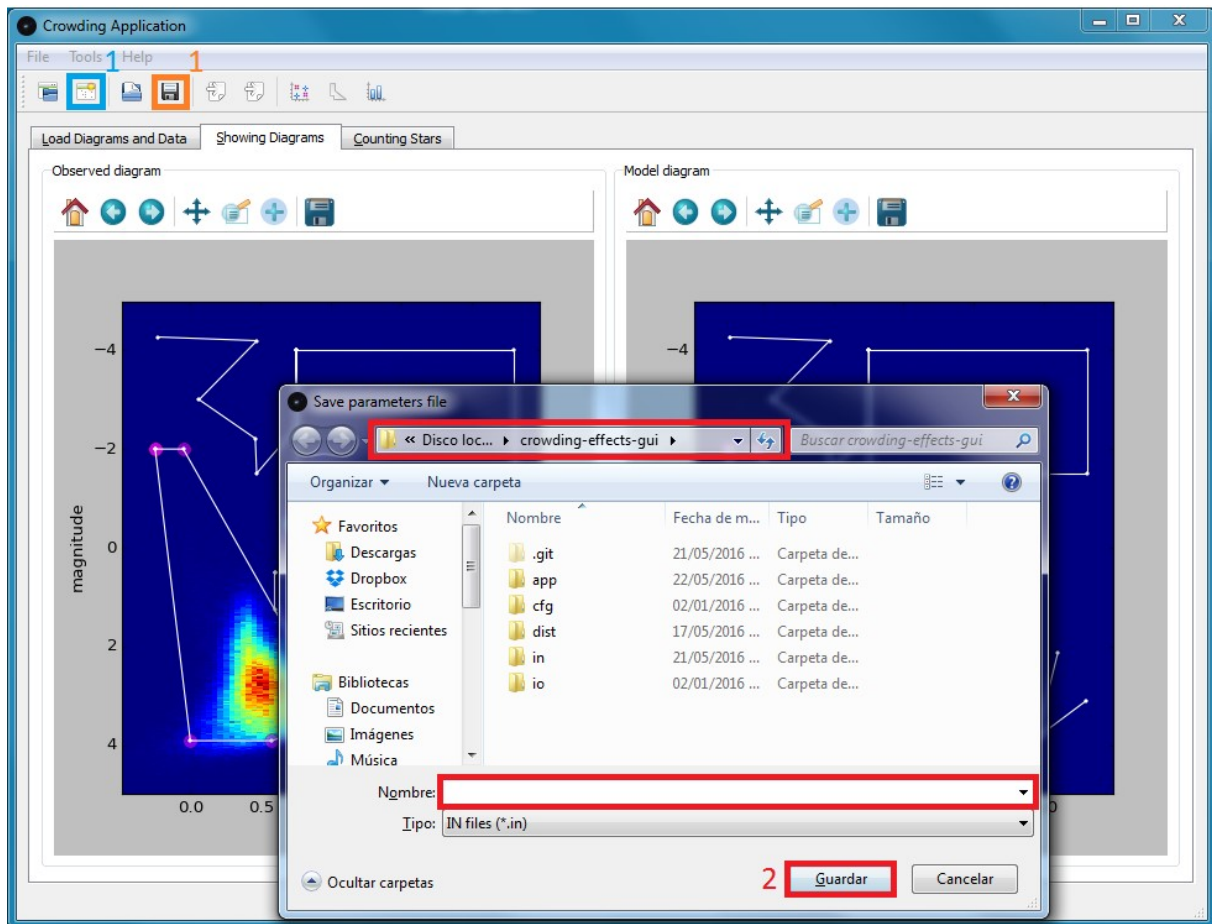


Figura 10.64: Ejemplo de uso de *Salvar el estado de la aplicación*.

Los pasos a seguir son:

1. Elegir la forma de salvar el estado de la aplicación.
 - a) Seleccionando el icono (1) para desplegar el cuadro de diálogo que permite guardar ficheros con formato `.in`.
 - b) Seleccionando el icono (1) para desplegar el cuadro de diálogo que permite guardar ficheros con formato `.ini`.
2. Navegar por el árbol de directorios del sistema, hasta el directorio deseado, introducir un nombre para el fichero y pulsar en **Guardar**.

Este proceso puede ser ejecutado en cualquier momento de la ejecución de la aplicación.

10.5.3.13 Restaurar el estado de la aplicación

La figura 10.65 muestra el proceso que debe seguir el usuario para restaurar el estado de la aplicación. Se puede restaurar el estado de la aplicación de dos formas, mediante ficheros `.in` o ficheros `.ini`.

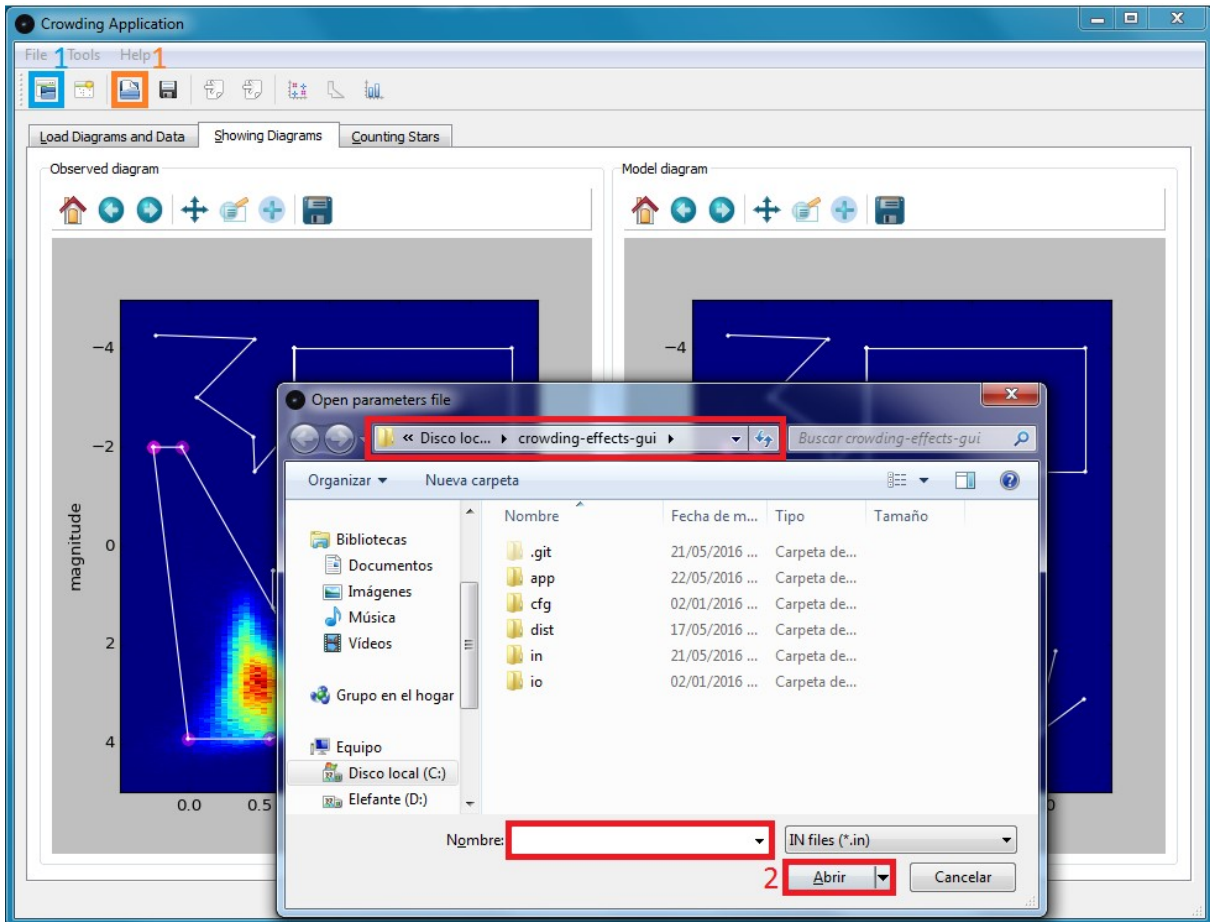


Figura 10.65: Ejemplo de uso de *Restaurar el estado de la aplicación*.

Los pasos a seguir son:

1. Elegir la forma de restaurar el estado de la aplicación.
 - a) Seleccionando el icono (1) para desplegar el cuadro de diálogo que permite restaurar ficheros con formato `.in`.
 - b) Seleccionando el icono (1) para desplegar el cuadro de diálogo que permite restaurar ficheros con formato `.ini`.
2. Navegar por el árbol de directorios del sistema, hasta el directorio donde se encuentra el archivo, seleccionarlo y pulsar en **Abrir**.

Este proceso puede ser ejecutado en cualquier momento de la ejecución de la aplicación.

10.5.3.14 Contar estrellas en rangos de edad y metalicidad

Las figuras que se muestran a continuación ilustran el proceso que debe seguir el usuario para poder contar el número de estrellas que hay en los rangos de edad y metalicidad.

La figura 10.66 muestra las expresiones que deben estar cargadas en la pestaña *Load Diagrams and Data* para poder calcular el número de estrellas que hay en los rangos de edad y metalicidad:

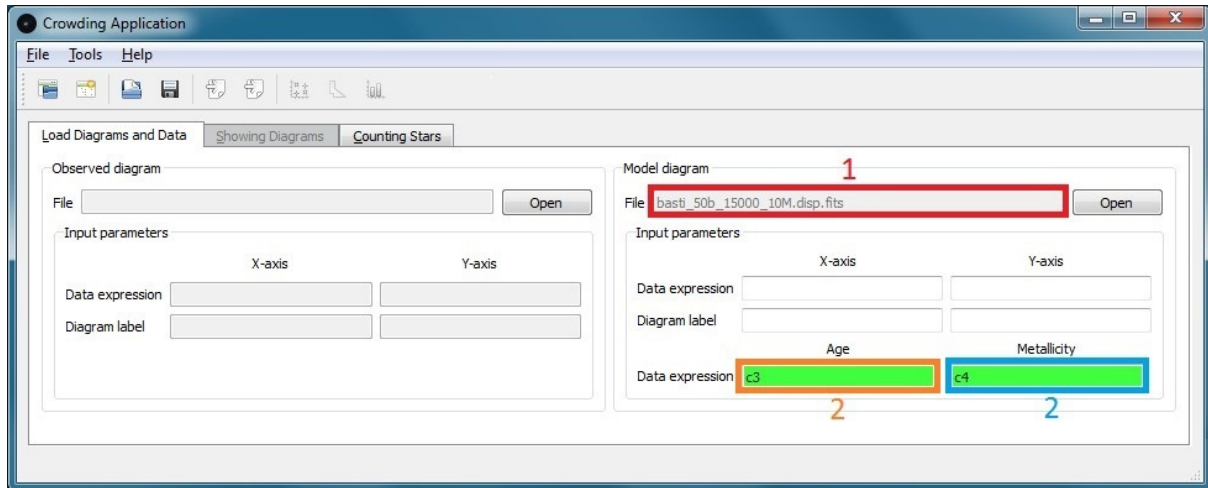


Figura 10.66: Ejemplo de uso de *Contar estrellas en rangos de edad y metalicidad*, expresión.

Primero se debe seguir el siguiente proceso:

1. Cargar el fichero correspondiente al diagrama modelo.
2. Cargar las expresiones de edad y metalicidad:
 - a) Para calcular el número de estrellas que hay en los rangos de *edad* se debe cargar la expresión (2).
 - b) Para calcular el número de estrellas que hay en los rangos de *metalicidad* se debe cargar la expresión (2).

La figura 10.67 ilustra los últimos pasos que deben llevarse a cabo para calcular el número de estrellas que hay en los rangos de edad y metalicidad.

Finalmente para terminar con el proceso:

3. Situarse en la pestaña *Counting Stars*.
4. Introducir los rangos para los que se quiere calcular el número de estrellas:
 - a) Para calcular el número de estrellas que hay en los rangos de *edad* se debe cargar la línea (4).
 - b) Para calcular el número de estrellas que hay en los rangos de *metalicidad* se debe cargar la línea(4).

Los valores introducidos en las líneas serán tomados de dos en dos (repetiendo el último valor tomado) y constituirán los distintos rangos para los que se realizará el recuento.

5. Los rangos de edad y metalicidad junto con el número de estrellas se mostrarán en las tablas.

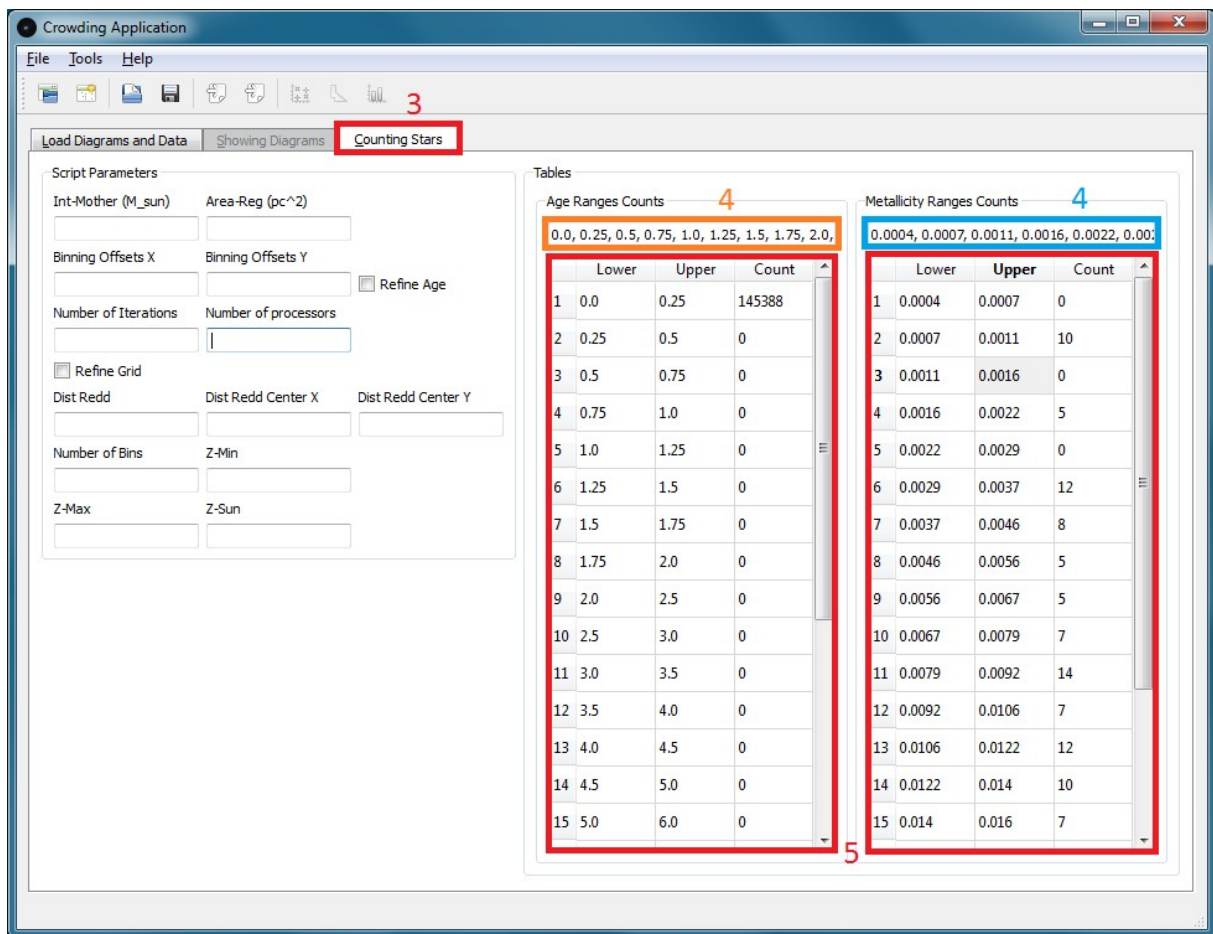


Figura 10.67: Ejemplo de uso de *Contar estrellas en rangos de edad y metalicidad*, rangos.

Parte III

Conclusiones y líneas futuras

Capítulo 11

Conclusiones y líneas futuras

En este capítulo se mencionan las conclusiones y experiencias adquiridas a lo largo de la realización de este proyecto. Además se revisan las posibles mejoras que pueden aportar un valor añadido al trabajo realizado, ya que toda aplicación es susceptible a ser mejorada.

11.1 Conclusiones

La realización de este proyecto nos ha hecho conscientes de la vital importancia que tienen, en un ámbito real, el empleo de técnicas y metodologías sistemáticas, relacionadas con la ingeniería del software, y nos ha permitido aprender que es necesario utilizar herramientas que permiten realizar un desarrollo más rápido y eficiente. Algunas de estas metodologías y herramientas son las siguientes:

- Adquisición de requisitos.
- Definición de los objetivos.
- Desarrollo ágil como *Scrum*.
- Entornos de desarrollo integrados (IDE).
- Control de versiones distribuido como Git.
- Automatización de tareas como *Ant*, *Maven* y *CMake*.

Emplear estas metodologías correctamente y utilizar estas herramientas desde el inicio, nos hubiera supuesto un ahorro significativo en tiempo y esfuerzos.

Sin embargo, esta experiencia nos ha servido para valorar positivamente la planificación previa al desarrollo y todo lo aprendido en los años de estudio académico previos. En definitiva, esta experiencia ha sido enriquecedora y una preparación para nuestro futuro en la vida laboral.

11.2 Líneas futuras

En este proyecto se han abordado múltiples problemas y múltiples implementaciones sobre un mismo problema, cada una de las cuales ofrece ventajas según las necesidades del grupo de investigación '*Evolución de galaxias en el Grupo Local*', del Instituto Astrofísico de Canarias. Este hecho determina que las posibles mejoras que se pueden añadir a lo desarrollado en este proyecto, se enumeren por separado.

Las implementaciones elaboradas en este proyecto se pueden agrupar atendiendo al tipo de interfaz de usuario:

11.2.1 Interfaz de línea de comandos

Un de las *líneas de investigación* que los futuros desarrolladores pueden explorar sobre las implementaciones realizadas, es plantearse utilizar otras técnicas de *Big Data*, como son: *MapReduce*, *HyperTable*, *Clustering*, etcétera. Esta línea de investigación supondría la reestructuración casi total del problema planteado en este proyecto.

11.2.1.1 Implementación recursiva con paralelismo sobre CPU

En esta implementación las mejoras que se pueden realizar son las siguientes:

- Utilizar alguna librería de representación aritmética de números en coma flotante que permita mejorar las funciones de comparación de números reales, para que trabajen más rápidamente.

Una opción sería utilizar la *GNU Multiple Precision Arithmetic Library (GMP)*. Esta librería permite no establecer límites en la precisión de los números y los compara de forma eficiente. Se puede consultar más información en la dirección web de su página principal: <https://gmp1ib.org/>.

- Mayor flexibilidad para definir el cálculo de las columnas de las estrellas dispersadas.

Sirva como ejemplo las siguientes líneas del fichero de parámetros en formato JSON donde las variables del atributo `columns` tienen el siguiente significado

- `r#`: columna # de la estrella real utilizada para la búsqueda.
- `a#`: columna # de la estrella artificial seleccionada aleatoriamente de entre las que cumplen todas las condiciones de búsqueda.
- `loop`: número de búsquedas realizadas para la estrella real.
- `size`: cantidad de estrellas artificiales que cumplen todas las condiciones de búsqueda.

```
{
  "DispersedStars" : {
    "file"      : "./files/dispStars.databs",
    "columns"   : [
      "r1-(a1 - a3)",
      "r2-(a2 - a4)",
      "r3",
      "r4",
      "r5",
      "r6",
      "loop",
      "size"
    ]
  }
}
```

- Utilizar librerías más estándar para calcular el resultado de las expresiones matemáticas que definen los índices de búsqueda.

Una opción puede ser *GNU libmatheval*. Para más información sobre esta librería, se puede consultar en su página web:

<https://www.gnu.org/software/libmatheval/>.

- Soporte para la ejecución en entornos distribuido de altas prestaciones, mediante el uso de *Open Message Passing Interface (OpenMPI)*.

Añadir esta funcionalidad, permitirá ejecutar en entornos de supercomputación como en el superordenador *Teide-HPC (High Performance Computing)*.

Para más información sobre *OpenMPI*, se puede consultar su página web oficial en: <https://www.open-mpi.org/>.

11.2.1.2 Implementación: Paralelismo a nivel de CPU y de GPU

En la implementación explicada en el capítulo 8 se pueden introducir las siguientes mejoras:

- Mejorar la comparación de números reales mediante el uso de la librería *GMP*.
- Se ha usado paralelismo a nivel de CPU mediante OpenMP. La librería *OpenMPI* permite usar paralelismo en entornos de supercomputación como ya se ha mencionado anteriormente.
- Realizar los cambios necesarios en el código de la implementación para poder usar cualquier tipo de GPU independientemente de la marca. Para ello se debe hacer uso del estándar OpenACC. Para más información sobre OpenACC se puede consultar la URL <http://www.openacc.org/>.
- Introducir los cambios necesarios que permitan usar todas las GPUs que hay disponibles en el sistema.
- Desarrollar las funciones recursivas que crean la estructura índice AVL y realizan la transformación a índice vector, incluyendo funciones de búsqueda iterativa sobre la estructura índice vector. De esta forma aumentaríamos la versatilidad de la implementación sin desperdiciar tiempo con búsquedas recursivas.
- Por último un desarrollador que se haya leído el capítulo 8 podría introducir algunos ficheros de código más que permitirían realizar búsquedas en base a otro número de características.
 - Basándonos en el contenido de los ficheros que se muestran a continuación crearíamos otros ficheros para el número de características deseadas.
 - `indice_avl4.c`
 - `indice_vector4.c`
 - `dispersion4.c`
 - `salida_crowding4.c`
 - `tipos4.h`
 - Habría que introducir cambios menores en los ficheros de código:
 - `tipos.h`
 - `constantes.h`
 - `crowd_main.c`
 - `parametros_entrada.c`

11.2.2 Interfaz gráfica de usuario

Las mejoras que se pueden añadir en la aplicación con interfaz gráfica pueden ser las siguientes:

- Ofrecer soporte para un número mayor de ficheros.
- Añadir funcionalidades propias del tratamiento de imágenes sobre los diagramas observado y modelo, que permitan una mejor definición de los bundles.
- Evitar la pérdida de cifras significativas a partir del quinto dígito decimal.

Esto afecta principalmente, a la creación de un nuevo punto que sea la intersección entre una línea definida y el punto anterior (ver Botón izquierdo del ratón sobre una recta). Al hacer zoom sobre el nuevo punto en cualquiera de los diagramas color-magnitud, se aprecia que el punto no intercepta la recta. Esto se debe a la pérdida de cifras significativas.
- Añadir nuevas vistas, acciones o funcionalidades según la necesidad del grupo de investigación ‘*Evolución de galaxias en el Grupo Local*’, del Instituto Astrofísico de Canarias.

Bibliografía

- Aparicio, Antonio y Carme Gallart. «IAC-STAR: A Code for Synthetic Color-Magnitude Diagram Computation». En: *The Astronomical Journal* 128.3 (2004), pág. 1465 (vid. págs. 4, 87).
- Continuum Analytics, Inc. *Anaconda documentation*. English. Continuum Analytics, Inc. URL: <https://www.continuum.io/documentation> (visitado 04-2016).
- Dunn, Christopher. *JsonCpp, a C++ library for interacting with JavaScript Object Notation (JSON)*. English. GitHub, Inc. URL: <https://github.com/open-source-parsers/jsoncpp/releases/tag/1.4.4> (visitado 04-2016).
- Instituto Astrofísico de Canarias. *Desarrollo del Programa público IAC-STAR*. Spanish. Instituto Astrofísico de Canarias. URL: <http://www.iac.es/proyectos.php?op1=40&y=2004&id=18> (visitado 04-2016) (vid. pág. 4).
- ISO. *ISO/IEC 14882:2011 Information technology — Programming languages — C++*. Geneva, Switzerland: International Organization for Standardization, 28 de feb. de 2012, 1338 (est.) (Vid. págs. 30, 33, 34).
- JetBrains S.R.O. *PyCharm*. English. JetBrains S.R.O. URL: <https://www.jetbrains.com/pycharm/> (visitado 04-2016).
- NVIDIA Corporation. *CUDA Toolkit Documentation v7.5*. English. NVIDIA Corporation. URL: <http://docs.nvidia.com/cuda/index.html> (visitado 04-2016).
- *GPU Accelerated Computing with C and C++*. English. NVIDIA Corporation. URL: <https://developer.nvidia.com/how-to-cuda-c-cpp> (visitado 04-2016).
- *Procesamiento paralelo CUDA Qué es CUDA*. Spanish. NVIDIA Corporation. URL: <http://www.nvidia.es/object/cuda-parallel-computing-es.html> (visitado 04-2016).
- OpenMP Architecture Review Board. *OpenMP Specifications*. English. OpenMP Architecture Review Board. URL: <http://openmp.org/wp/openmp-specifications/> (visitado 04-2016).
- Parr, Terence. *ANother Tool for Language Recognition (ANTLR)*. English. University of San Francisco. URL: <http://www.antlr3.org/download.html> (visitado 04-2016) (vid. pág. 45).
- Partow, Arash. *The C++ Mathematical Expression Toolkit Library (ExprTk)*. English. GitHub, Inc. URL: <https://github.com/ArashPartow/exprtk> (visitado 04-2016).
- Python Software Foundation. *Python 2.7.11 documentation*. English. Python Software Foundation. URL: <https://docs.python.org/2/> (visitado 04-2016).
- *Python Developer's Guide*. English. Python Software Foundation. URL: <https://docs.python.org/devguide/> (visitado 04-2016).

-
- *Python information in Spanish*. Spanish. Python Software Foundation. URL: <https://wiki.python.org/moin/SpanishLanguage> (visitado 04-2016).

 - Riverbank Computing Limited. *PyQt4 Reference Guide*. English. Riverbank Computing Limited. URL: <http://pyqt.sourceforge.net/Docs/PyQt4/> (visitado 04-2016).

 - *What is PyQt?* English. Riverbank Computing Limited. URL: <https://www.riverbankcomputing.com/software/pyqt/intro> (visitado 04-2016).

 - The Qt Company Ltd. *Qt Documentation, Qt 4.8, All Classes*. English. The Qt Company Ltd. URL: <http://doc.qt.io/qt-4.8/classes.html> (visitado 04-2016).

 - *Qt Documentation, Qt 4.8, UI Design with Qt*. English. The Qt Company Ltd. URL: <http://doc.qt.io/qt-4.8/qt-gui-concepts.html> (visitado 04-2016).

Acrónimos

A | C | E | G | H | I | J | N | P | R | T | U | V

A

ANTLR ANother Tool for Language Recognition.

API Application Programming Interface.

AVL Adelson-Velsky Landis.

C

CDT C/C++ Development Toolkit.

CLI Command Line Interface.

CPU Unidad de Procesamiento Central.

E

ExprTk The C++ Mathematical Expression Toolkit Library.

G

GCC GNU Compiler Collection.

GNU GNU's Not Unix.

GPU Unidad de Procesamiento Gráfico.

GUI Graphical User Interface.

H

HTML HyperText Markup Language.

I

IAC Instituto Astrofísico de Canarias.

IDE Integrated Development Environment.

IEC International Electrotechnical Commission.

IEEE Institute of Electrical and Electronics Engineers.

IETF The Internet Engineering Task Force.

ISO International Organization for Standardization.

J

JSON JavaScript Object Notation.

N

NVCC NVIDIA's CUDA Compiler.

P

PC Personal Computer.

PFC Proyecto de Fin de Carrera.

R

RAM Random Access Memory.

RFC Request for Comments.

T

TFC Trabajo de Fin de Carrera.

U

ULL Universidad de La Laguna.

UML Unified Modeling Language.

V

VRAM Video Random Access Memory.

Glosario

A | C | D | E | F | G | H | J | K | L | M | O | P | Q | S | T | U | V

A

Adelson-Velsky Landis es un tipo especial de árbol binario ideado por los matemáticos rusos Adelson-Velskii y Landis.

ANother Tool for Language Recognition es un Framework que genera el código fuente para diferentes lenguajes de programación de analizadores, intérpretes, compiladores y traductores a partir de la descripción de una gramática. Para más información, consultar la página web: <http://www.antlr.org/>.

Application Programming Interface es un conjunto de rutinas, incluyendo clases, estructuras, funciones, métodos, etc., que detalla las funcionalidades para un soporte extendido e integrado de aplicaciones.

C

C es un lenguaje de programación de alto nivel. En sus inicios fue desarrollado por Dennis M. Ritchie entre 1969 y 1972 en los Laboratorios Bell. Es un lenguaje muy apreciado por la eficiencia del código producido y es usado en la actualidad para crear software de sistemas y aplicaciones..

C++ es un lenguaje de programación de propósito general de nivel intermedio basado en C. **C++11** es el nombre del estándar de C++, aprobado en 2011. Añade varios cambios en el núcleo del lenguaje, la mejora en la manipulación de arreglos y una biblioteca estándar mejorada y ampliada.

CUDA es la arquitectura de computación paralela de NVIDIA que aprovecha la gran potencia de la GPU proporcionando un incremento sustancial del rendimiento del sistema..

CUDA Toolkit es una herramienta de NVIDIA que proporciona un entorno de desarrollo completo para que los desarrolladores de C/C++ implementen aplicaciones aceleradas por Unidades de Procesamiento Gráfico (GPUs) NVIDIA. Incluye un compilador para las GPUs NVIDIA, librerías matemáticas y herramientas para depurar y optimizar el rendimiento de las aplicaciones..

D

Doxygen es una aplicación que genera documentación para múltiples lenguajes de programación como: C++, C, Java, Objective-C, Python, IDL, Fortran, VHDL, PHP y C#. Además, soporta una amplia variedad de formatos de salida como: RTF, HTML, XML, JSON, PDF y \LaTeX .

E

Eclipse es software multiplataforma, categorizado como un IDE, que permite el desarrollo de aplicaciones para diferentes lenguajes de programación. Además, posee un amplio abanico de plugin opcionales, que lo hace más atractivo para los desarrolladores. Para más información se puede consultar su página web en: <http://www.eclipse.org/>.

F

Fortan es un lenguaje de programación de alto nivel. Su uso esta orientado cálculo numérico y a la computación científica..

Framework es una herramienta que estandariza y ofrece una infraestructura sobre un campo determinado, en el argot de la informática.

G

Git es un sistema de control de versiones distribuido, gratuito y de código abierto. Para una información más detallada consultar: <https://git-scm.com/>.

GNU/Linux es una término que se emplea para referirse al sistema operativo *GNU* combinado con el núcleo o kernel Linux.

Google C++ Testing Framework es una librería de pruebas unitarias para C++, desarrollada por *Google* y basada en la arquitectura *xUnit* Para una información más detallada consultar: <https://github.com/google/googletest/>.

H

HyperText Markup Language es el lenguaje de marcado estándar usado para la estructuración de páginas web y formato de contenido.

J

Java es un lenguaje de programación de propósito general. Para ejecutar requiere una Máquina Virtual de Java (JVM).

JavaScript Object Notation es un formato de intercambio destinado a ser legible por el ordenador y el humano. Fue definido por The Internet Engineering Task Force (IETF) en el **Request for Comments** (RFC) 7159 que es completamente independiente del lenguaje. Para más información, consultar la página web: <http://www.json.org/>.

K

Kubuntu es una distribución Linux que usa KDE como entorno de escritorio. Ha sido desarrollado por Blue Systems y sus colaboradores. Es una distribución que deriva de la distribución Ubuntu..

L

Linux es un sistema operativo, descendiente de UNIX, desarrollado bajo el modelo de software libre y abierto (Free and Open Source Software).

M

Makefile es normalmente un fichero de entrada para la herramienta de control de compilación **make**. Para obtener más información, se puede consultar: <https://www.gnu.org/software/make/>.

O

OpenACC es un estándar de programación para computación paralela desarrollado por Cray, CAPS, Nvidia y PGI. Este estándar ha sido diseñado para simplificar la programación paralela de sistemas heterogéneos..

OpenCL es un estándar abierto multiplataforma para la programación paralela de los diversos procesadores con los que nos podemos encontrar en ordenadores personales, servidores, dispositivos móviles y sistemas embebidos..

OpenMP es una API multiplataforma orientada al multi-threading, que permite la paralelización y la sincronización de tareas, mediante el uso de directivas especiales del compilador. Ofrece una fácil manipulación de múltiples hilos sin requerir conocimiento de detalles dependientes del sistema. Para una información más detallada consultar: <http://openmp.org/>.

P

PyQt es un binding de la biblioteca gráfica Qt para el lenguaje de programación Python.

Python es un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma..

Q

Qt es una biblioteca multiplataforma para C++ usada tanto para el desarrollo de aplicaciones, ya sean de escritorio, para móvil o tableta, consola de comandos, etcétera.

S

Swap referido al proceso por el que una página de memoria se copia en un espacio del disco configurado previamente para liberar la memoria RAM. También recibe el nombre de área de intercambio.

T

The C++ Mathematical Expression Toolkit Library es una librería para C++ que permite analizar y evaluar de expresiones matemáticas en tiempo de ejecución. Para más información, consultar la página web:
<https://github.com/ArashPartow/exprtk/>.

U

Ubuntu es un sistema operativo de servidor y de escritorio gratuito basado en Debian GNU/Linux.

Unified Modeling Language es una lenguaje de modelado estándar de propósito general fuertemente relacionado con el campo de la ingeniería de software.

V

Valgrind es un conjunto de herramientas orientadas a la depuración de problemas de memoria y rendimiento de programas. Para una información más detallada consultar: <http://valgrind.org/>.

Índice alfabético

- análisis
 - léxico, 29, 30, 32, 44, 45
 - sintáctico, 29, 30, 32, 44, 45
- ANTLR, 45
- árbol, 41
 - AVL, 36, 37
 - anidado, 37
 - balanceado, 36, 37
 - directorio, 35, 94
- bug, 30, 32, 35, 45
- bundle, 87–89, 98, 99, 101, 107–111, 115, 117, 118, 121, 122, 125, 127–133, 135, 145
 - abierto, 118, 122
 - cerrado, 118, 130, 135
 - condiciones, 118, 132
- búsqueda, 37, 50
 - binaria, 37
- característica, *véase* índice
- color, 87
- cota superior, 36, 37
- CPU, 29
- diagrama, 101, 107–110, 114, 115, 125
 - color-magnitud, 87–89, 98–100, 114, 115, 117, 118, 122–124, 134, 145
 - Hess, 87
 - modelo, 88, 101, 107, 114, 115, 135, 138, 145
 - observado, 87, 88, 107, 114, 115, 135, 145
 - sintético, 87, 88
- dispersión, 38, 39
- documentación, 44
- edad, 87–89, 101, 114, 116, 138
- ejecución
 - abortada, 44
 - depuración, 46
 - estéril, 44
 - lanzamiento, 46
 - parámetro, 47
 - prueba, 46
- error, 30, 44, 45, 47
 - margen, 39, 42
 - sintáxis, 44
- estrella
 - artificial, 47
 - artificial, 29, 38, 39, 42, 44, 46, 47, 49, 50
 - conjunto, 38
 - crowding, *véase* artificial
 - dispersada, 39, 44, 47, 50, 144
 - madre, *véase* real
 - real, 29, 38, 39, 42, 44, 46, 47, 50, 51
 - sintética, *véase* real
- estructura, 36, 49
 - árbol, 41, 43
 - anidada, 40, 43, 44
 - lista, 41, 43
- expresión, 120
 - evaluador, 29
 - matemática, 29, 44, 45, 47, 114, 138
- fichero
 - artificial, 45, 47–49
 - dispersado, 44, 47, 48
 - ejemplo, 44
 - entrada, 44
 - inexistente, 44
 - madre, 44
 - ordenado, 44, 47
 - parámetro, 36, 40–42, 44–48, 50, 88, 144
 - real, 45, 47–49
 - registro, 29, 30, 45, 48
- histograma, 89, 100, 108, 118, 135
- índice, 36, 41, 42
 - anidado, 36, 37, 39–42, 44
 - búsqueda, 36, 39, 44, 47
- JSON, 29, 47
- lista, 36, 37, 42, 44
 - anidada, 37
- magnitud, 87
- metalicidad, 87–89, 101, 114, 116, 138
- nivel, *véase* índice
- nodo
 - derecha, 40
 - izquierda, 40
 - padre, 40
 - primero, 41
 - raíz, 41
 - siguiente, 40
 - último, 41

- OpenMP, 29
- proceso
 - análisis, 45
 - búsqueda, 30, 44, 46
 - desarrollo, 46
 - dispersión, 38, 39, 42, 44, 46, 48, 50–52
 - dispesión, 50
 - lote, 44
- profundidad, 40–42, 44
- RAM, 29
- recursividad, 45
- resultado, 50
- script, 88, 89
- target, 32–34, 36
 - Debug, 33
 - Release, 34
 - Test, 34
- variable
 - export, 33