TÍTULO

ANÁLISIS BIOINFORMÁTICO DE VÍAS DE TRANSDUCCIÓN HUMANAS CON IDENTIFICACIÓN DE PARÁLOGOS EN CONTEXTO TRANSCRIPTÓMICO

AUTOR

Daniel Pérez Martín

Esta edición electrónica ha sido realizada en 2012

Director	Javier de las Rivas
Tutor	Toni Gabaldón
Curso	Máster Oficial en Bioinformática
©	Daniel Pérez Martín
©	Para esta edición, la Universidad Internacional de Andalucía

Universidad Internacional de Andalucía, 2012

Reconocimiento-No comercial-Sin obras derivadas

Usted es libre de:

- Copiar, distribuir y comunicar públicamente la obra.

Bajo las condiciones siguientes:

- **Reconocimiento**. Debe reconocer los créditos de la obra de la manera. especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).
- **No comercial**. No puede utilizar esta obra para fines comerciales.
- **Sin obras derivadas**. No se puede alterar, transformar o generar una obra derivada a partir de esta obra.

- *Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.*
- *Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor.*
- *Nada en esta licencia menoscaba o restringe los derechos morales del autor.*

Universidad Internacional de Andalucía

Máster en Bioinformática

Trabajo fin de máster

ANÁLISIS BIOINFORMÁTICO DE VIAS DE TRANSDUCCIÓN HUMANAS CON IDENTIFICACIÓN DE PARÁLOGOS EN CONTEXTO TRANSCRIPTÓMICO

Autor: Daniel Pérez Martín

Dirección: Javier de las Rivas y Toni Gabaldón.

León, 26 de marzo de 2012

Dedico este trabajo a todo aquel que lo lea.

1. ÍNDICE GENERAL

ÍNDICE GENERAL

2. INTRODUCCIÓN

2.1. Motivación

El procesamiento de los datos complejos es un reto para la Bioinformática hoy en día. Asimismo su representación suele presentarse como un problema añadido. Existen múltiples ejemplos de esto, entre los que se encuentra la representación de las vías metabólicas y de señalización, y la integración de la información que subyace en ellas. Una de las soluciones a esta cuestión, como propone la base de datos KEGG, es la de descomponer el conjunto de relaciones en elementos simples [1]. Una de las mayores ventajas de este planteamiento, estriba en que las unidades resultantes pueden manejarse con facilidad en el ámbito computacional al tratarse de entidades simples.

La información contenida en cualquier vía metabólica está en continua revisión, puesto que se trata de información experimental, que cambia con los avances en Biología Molecular. Para hacer frente a la actualización de las vías se requieren estrategias que enlacen la representación con la información experimental. En concreto KEGG asocia genes a los nodos representados en las vías. No obstante muchos de los nodos se encuentran asociados no a un solo gen, sino a una familia de genes o conjunto de genes parálogos. En estos casos, las relaciones asociadas a dichos nodos se referirán a cualquiera de los genes de la familia. Esto plantea una posible mejora de la vía: el establecer las relaciones entre los nodos asociados a familias de genes (familia-a-familia) a un nivel de detalle mayor, es decir, genes concretos (gen-a-gen). Esto supone un incremento en la información contenida inicialmente en la vía representada, por lo que exige nuevas fuentes de datos para generar dicha información.

Los microarrays de rango genómico son una técnica usual en estudios de expresión de genes actualmente. De estos podemos extraer información de coexpresión, que indica la activación conjunta de genes. Esta coexpresión significa que estos genes interactúan de algún modo, por lo que también puede enriquecer la información de relaciones representadas en las vías KEGG.

La secuenciación de múltiples genomas en los últimos años, permite elevar la comparación de secuencias a un nivel mayor. Una de las posibilidades que se abren con esto es la reconstrucción filogenética de cladogramas de genes a gran escala, de manera que nace una nuevo concepto análogo al genoma: el filoma, que no es sino el conjunto de reconstrucciones filogenéticas de todos los genes de una especie [2]. Los genes en estos

filomas se relacionan con otros de su especie, pero también con genes de otras especies. PhylomeDB es una base de datos que almacena estos filomas, y por tanto, información sobre la evolución de genes en un contexto de múltiples genomas [3]. Estos datos de evolución pueden extraerse y compararse gen a gen para inferir una coevolución que indicaría una relación entre estos dos genes. Esta relación también se puede corresponder con las que ofrece KEGG, de modo que es susceptible de ser una fuente de datos adicional para integrar su información en las vías KEGG.

Se puede acceder a cada uno de estos conjuntos de datos (KEGG, datos de expresión, PhylomeDB) de forma programática, por lo que es posible diseñar un sistema que integre la información contenida en una y otras fuentes para lograr representar mayor información en las vías metabólicas y de señalización presentes en KEGG.

La **hipótesis de trabajo** es que existen preferencias de relación entre los subelementos (genes parálogos) de los nodos representados en KEGG, es decir, un subelemento se relaciona preferentemente con otro, y que estas preferencias se pueden establecer consultando otras fuentes de información, como son datos de expresión y de evolución de los genes implicados.

2.2. OBJETIVOS

- **Análisis de las relaciones entre familias de genes presentes en la vía de las MAPK en KEGG a nivel de genes concretos mediante coexpresión y coevolución.**
- Extracción del conocimiento de relación entre los genes de interés mediante el análisis de su coexpresión en microarrays.
- Extracción del conocimiento de relación entre los genes de interés mediante el análisis de su coevolución a partir de los datos contenidos en la base de datos PhylomeDB.
- Diseño y programación de una plataforma para facilitar el análisis de los datos de interés y permitir su ampliación en el futuro.

3. SITUACIÓN ACTUAL

Con el desarrollo de las tecnologías de la información y la comunicación, se han abierto múltiples posibilidades en la interpretación y compartición de los resultados experimentales en biología. En la mayoría de los casos estos resultados no están codificados de una forma apropiada para su consulta o procesamiento por parte de programas informáticos, por lo que se requiere una organización manual [4]. Esta organización abarca múltiples tipos de información, como puede ser la anotación de genes o la reconstrucción de vías metabólicas y de señalización.

Una vez que existe una base de datos que contiene información se puede acceder a ella mediante redes telemáticas como Internet y/o procesada por programas informáticos. En el caso de las vías metabólicas y de señalización, la información puede ser consultada desde la más general (una vía) o desde la más particular (un gen o molécula) [1]. El análisis de vías es una buena estrategia para el entendimiento de la biología subyacente en los experimentos de expresión de genes, puesto que reduce la complejidad y posee un potencial explicativo mayor con respecto a otros abordajes [5].

Los datos generados en la era de la secuenciación genómica crecen de forma exponencial. Este hecho aporta tanto problemas de organización como oportunidades de incremento del conocimiento. A su vez, abordar ambos exige el desarrollo de nuevos algoritmos informáticos para la minería de datos y extracción del conocimiento. La disponibilidad de diversos genomas completos abre la posibilidad de establecer las relaciones evolutivas de todos los genes de un organismo, reconstruyendo sus historias filogenéticas en comparación con los genes del mismo genoma y de otros genomas. Estas estrategias que tratan de procesar una gran cantidad de datos exigen una capacidad de proceso, que no ha sido posible abordar hasta el incremento reciente de capacidad por parte de los componentes informáticos [2].

Los experimentos de expresión de rango genómico brindan una gran cantidad de datos que pueden ser procesados para incrementar el conocimiento de las funciones biológicas de la especie a la que pertenece dicho genoma. Un mismo conjunto de datos puede ser utilizado repetidamente con diversas estrategias para responder a diferentes interrogantes. Estos datos pueden ser compartidos de forma pública en bases de datos de experimentos de microarrays [6].

4. MATERIAL Y MÉTODOS

4.1. Origen de los datos

Los datos que utiliza el presente trabajo tienen diversos orígenes, y los programas diseñados tienen por objeto integrar la información para obtener una visión combinada de la misma.

a)Datos de vías metabólicas de KEGG [1]

Es una base de datos que alberga relaciones binarias (uno a uno) entre entidades biológicas. Un ejemplo de lo que se puede representar con esto son las vías metabólicas, y se ha escogido procesar la vía de las MAPK. Este es el punto de partida del resto de procesamiento.

b)Datos de expresión génica del CIC [7]

Los datos de expresión provienen de una colección de análisis de muestras de tejidos humanos hechas con microarrays de expresión génica de ámbito genómico (es decir, que abarcan la mayoría de los genes conocidos del genoma). Los microarrays son de la firma *Affymetrix*, modelo HG-U133A. Son un total de 48 muestras de 24 tejidos u órganos humanos distintos –con dos réplicas biológicas de cada tejido– [7].

c)Datos de evolución de PhylomeDB [8]

Los datos de evolución son obtenidos de la base de datos PhylomeDB. Se componen de de árboles que representan la reconstrucción de las filogenias de genes de diferentes especies [8]. Este trabajo utiliza sólo la filogenia de genes humanos.

4.2. Estructura de los datos

a)KEGG

Los datos de relaciones en KEGG se recogen a través de la API pública que facilita esta base de datos. Estos datos se reciben en forma de dos listas:

- **Elementos**, que representan los nodos relacionados. Tienen la siguiente estructura (un diccionario en Python):

```
{'element_id': 34, 'type': 'gene', 'names': ['hsa:6654',
'hsa:6655'], 'components': []}
```

  - Este proyecto utiliza los atributos “element_id”, “type”, “names”

- **Relaciones**, que guardan información sobre el modo en que se relacionan los elementos anteriores. Tienen la estructura:

```
{'element_id2': 32, 'element_id1': 34, 'type': 'PPrel', 'subtypes':
[<SOAPpy.Types.structType item at 41606968>: {'type': '-->',
'element_id': None, 'relation': 'activation'}]}
```

  - Este trabajo utiliza los atributos: “element_id2”, “element_id1”.

b) Expresión

Los datos crudos de expresión provienen de microarrays de expresión de rango genómico cuyos resultados se han exportado a ficheros de texto plano separados por tabulaciones. Se organizan en cuatro ficheros:

- Fenotipos
- Nombres y traducción en ID de Affymetrix
- Datos de expresión de cada gen obtenidos con el algoritmo MAS5
- Datos de expresión de cada gen obtenidos con el algoritmo RMA

Los dos últimos ficheros de datos de la señal de expresión se estructuran en un fichero de texto plano delimitado por tabuladores en el que cada línea es una sonda de detección génica (gene probe-set). En la primera columna se encuentra la referencia a la sonda a la cual pertenecen los datos que hay en esa línea. En el resto de columnas se encuentran los datos leídos y normalizados para ese gen en los 24 tejidos diferentes (dos réplicas por cada uno) [7].

c) PhylomeDB

Se ha utilizado la versión descargable de esta base de datos que se encuentra en la URI ftp://phylomedb.org/phylomedb/. En concreto los ficheros:

- **all_id_conversions**: que contiene referencias cruzadas de los genes a otras bases de datos.

- **best_trees**: que contiene los árboles filogenéticos reconstruidos con el algoritmo que funciona mejor para cada gen. Para la especie humana se encuentran en la ruta relativa “phylomes/phylome_0001/”

4.3. Filtración de los datos

Se han seguido una serie de criterios de filtración de los datos con el objeto de desechar los elementos incompletos (por ejemplo, elementos cuyas referencias cruzadas no obtenían ninguna ID válida en una fuente de datos) o que no eran de interés para este trabajo (por ejemplo, casos en los que un elemento no represente un gen). Por tanto, del conjunto de datos que se tienen en cada uno de los orígenes, se han considerado procesables y de interés un subconjunto concreto en cada caso que se detalla a continuación.

a)KEGG

Debido a que la información de que disponemos en los datos de expresión y evolución se refieren a genes, el primer filtro establece que sólo se procesarán los elementos KEGG que representen genes:

```
element['type'] == 'gene'
```

Asimismo, como el interés del proyecto es establecer las relaciones entre subelementos de KEGG, para establecer estas se procesan solamente los elementos que tienen al menos dos subelementos:

```
len(element['names']) >= 2
```

b)Expresión

Los datos de expresión del CIC se recuperan a través de los nombres presentes en KEGG para cada subelemento. Estos nombres incluyen una serie de sinónimos que detalla la base de datos para cada gen en el atributo “gene name” (por ejemplo, para el gen *SOS1* se contemplan los nombres: *SOS1, GF1, GGF1, GINGF, HGF, NS4*).

c)PhylomeDB

Los datos de PhylomeDB se recuperan a través de las referencias cruzadas que facilita KEGG para cada subelemento. Al buscar cada referencia cruzada, obtenemos un identificador (ID) de gen de PhylomeDB, y con cada uno de estos ID obtenemos un árbol filogenético para dicho gen.

4.4. Procesamiento

Las parejas de subelementos KEGG (genes) relacionados son la unidad a procesar en este proyecto. Con cada familia de genes relacionadas (elementos KEGG) se establecen todas las posibles parejas (un permutación de todos los elementos). Cada una de estas parejas se procesará según los dos enfoques propuestos en los objetivos:

a)Coexpresión

Los datos de expresión para cada gen consisten en una lista que representa el grado de expresión de dicho gen. La estrategia metodológica en este caso consiste en el cálculo de los coeficientes de correlación de Pearson y Spearman de las dos listas de datos de los dos genes relacionados. Una mayor correlación (valores cercanos a +1) indica una coexpresión (expresión conjunta de los genes) mayor. Las correlaciones negativas (cercanas a -1) indican lo opuesto (un gen se expresa cuando el otro no y viceversa). Correlaciones cercanas a 0 indican que no existe relación.

b)Coevolución

Los datos de evolución para cada familia de genes consisten en un árbol filogenético que relaciona dichos genes. A partir de dichos árboles se pueden extraer subárboles que sólo representen a un gen concreto o un conjunto de genes relacionados. La metodología utilizada es la de hallar los subárboles mayores tales que cada uno represente a un solo gen de la familia de genes que se estudia.

Una vez calculados los subárboles, para cada par de genes relacionados, se calculará la distancia evolutiva (teniendo en cuenta la topología de los subárboles) mediante la herramienta TreeKO [9]. El resultado oscilará entre 0 (significando una topología idéntica) y 1 (completamente distinta).

4.5. Tecnologías utilizadas

PhylomeDB: una base de datos de filomas. Los filomas se construyen a partir de los proteomas de otras bases de datos. Un filoma se define como el conjunto de relaciones filogenéticas de cada secuencia proteínicia de un organismo. En conjunto tenemos la relación evolutiva de cada proteína (o, en general, gen) de un organismo en concreto. Entre otros se encuentra el filoma humano [2].

ETE2 (versión 2.1rev492): es una toolkit en Python [10] que permite:

- El manejo de árboles como estructuras de datos en Python.
- El manejo de la filogenia subyacente dentro de los árboles, incluyendo conceptos de edad, distancia, etc.
- La adición o sustitución de sus funciones para el manejo de esta clase de datos.
- El acceso remoto a la base de datos PhylomeDB mediante API.

TreeKO: es un programa Python que permite comparar árboles y comparar su distancia basándonse en su topología [9].

Datos de expresión del CIC: los datos de expresión provienen de los microarrays realizados para el artículo *Human Gene Coexpression Landscape: Confident Network Derived from Tissue Transcriptomic Profiles* [7].

R (versión 2.12.1): un lenguaje de programación con funciones estadísiticas integradas. Es muy utilizado para el cálculo de estadística en Bioinformática u otros ámbitos de investigación [11].

Rpy2 (versión 2.1.9): una interfaz entre Python y R que permite el uso de funciones y estructuras de datos de R a través de Python [12].

Python (versión 2.1.7+): un lenguaje de programación de fácil aprendizaje y adaptación y que permite una programación ágil y rápida [13].

ImageJ (versión 1.46c): un programa de procesamiento de imagen digital de código abierto que permite la programación de macros para el análisis de imagen [14].

4.6. Representación de los resultados

Como resultado se obtienen tablas de datos: una por cada relación entre dos familias de genes. Cada fila o columna representa un gen concreto (filas para una familia y columnas para la otra). El resultado para cada par de genes se encuentra en la intersección.

En el caso de la coexpresión se representa el coeficiente de correlación calculado. Cada familia de genes contará con dos tablas, puesto que se calculan dos coeficientes de correlación: el de Pearson y el de Spearman. En la coevolución se representará la distancia entre subárboles.

Para facilitar la lectura de los datos se han representado las matrices de forma gráfica con un código de colores. Se ha asignado el verde a la máxima identidad en cada caso (una correlación de +1 en coexpresión y una distancia de 0 en coevolución) y negro a la mínima

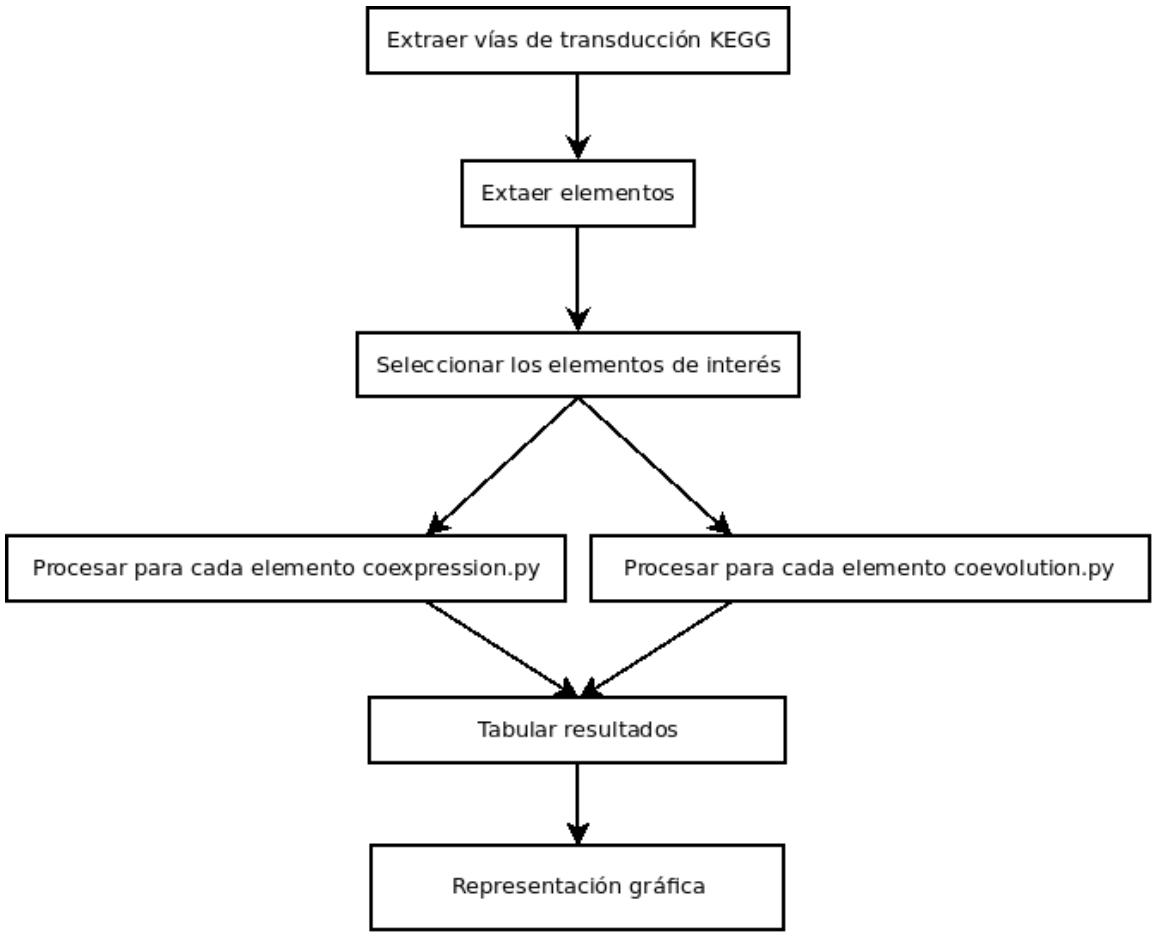
(0 y 1 respectivamente). Además en el caso de coexpresión, los coeficientes de correlación negativos se representan codificados asignando rojo al valor de -1.

Se ha utilizado un espacio de color RGB de 8 bit por canal, por lo que se calcula para cada resultado un valor entre 0 y 255 y se le asigna al canal verde o rojo según sea el caso. Para la obtención de estas representaciones de las matrices se ha diseñado una macro para el programa ImageJ [14].

Para integrar la información lo mejor posible, se han combinado los resultados numéricos tabulados con las matrices representadas gráficamente, es decir, se han construido tablas de resultados numéricos en las que cada celda tiene el fondo del color que corresponde a su resultado según la representación gráfica expuesta.

5. IMPLEMENTACIÓN

5.1. Diagrama de flujo general



Cada uno de estos pasos se explican con mayor detalle a continuación:

- Extraer vías KEGG: en este paso se descargan los datos de KEGG. Pueden descargarse los datos de cualquiera de las vías publicadas en KEGG a partir de su identificador. En el caso de la vía de las MAPK humana es “hsa04010”.
- Extraer los elementos: se establecen ambas listas, de elementos y de relaciones.
- Seleccionar elementos de interés: según los criterios de filtración anteriormente expuestos, se descartan los elementos que no sean de interés. (véase sección 4.3).
- Procesar cada elemento con coexpression.py: extrae los datos de expresión y calcula los coeficientes de correlación de Pearson y Spearman mediante R.

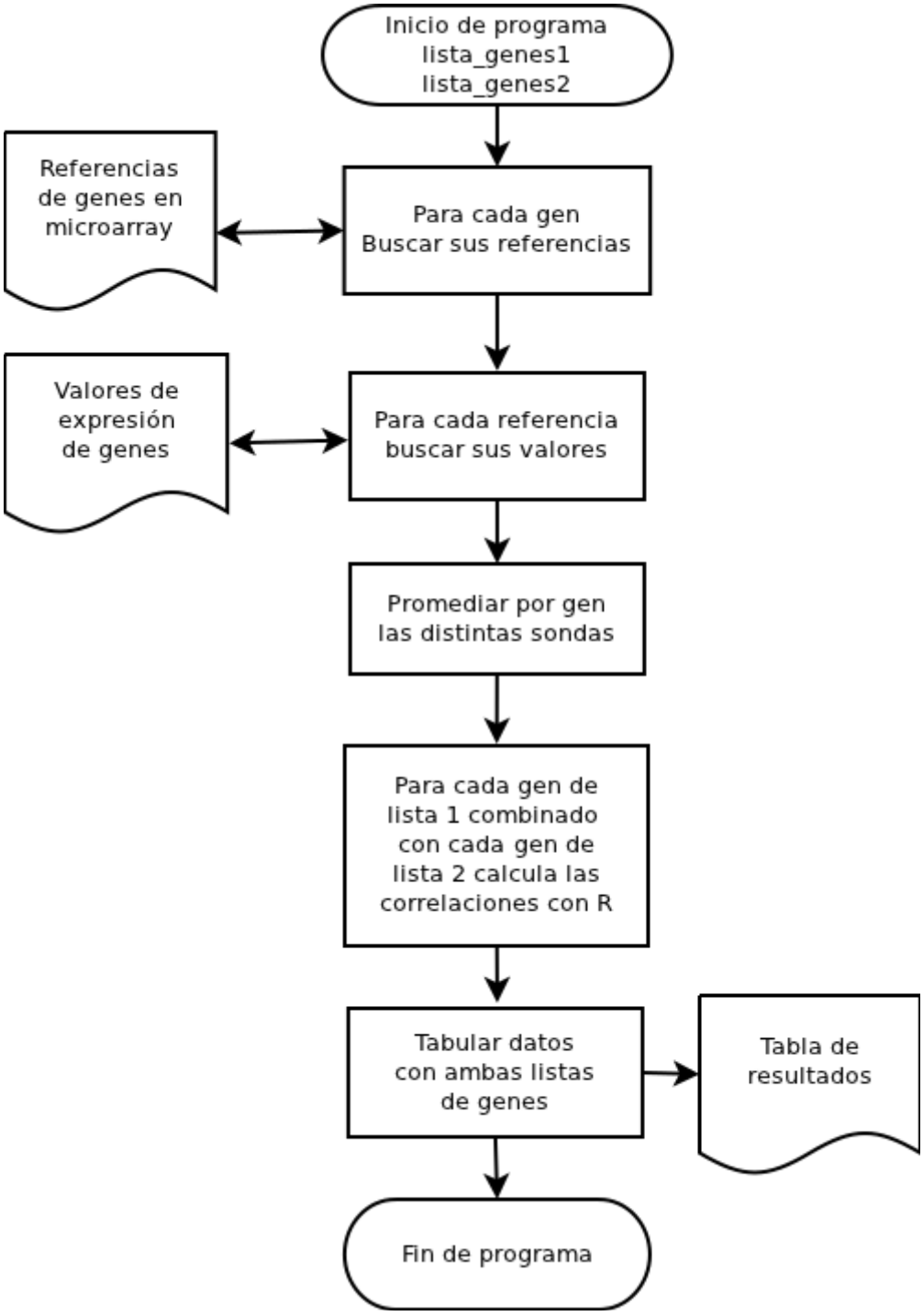
- Procesar cada elemento con coevolution.py: extrae los árboles filogenéticos para cada familias de genes, calcula los subárboles para las comparaciones y las realiza con TreeKO.

5.2. Extracción de los datos KEGG

El acceso a los datos de KEGG se realiza a través de su API pública (los datos de KEGG son definidos mediante el formato WSDL y accedidos mediante SOAP). A través de este sistema se puede tener acceso a cualquier información almacenada en KEGG. En concreto este trabajo recoge de KEGG la vía de las MAPK en el formato descrito en la sección 6.3. "*Estructura de los Datos*". Los datos se organizan en dos listas: una de relaciones y otra de elementos KEGG (o familias de genes). A partir de estos datos se generan instancias de las clases definidas en Python para abstraer la información de KEGG.

5.3. Programa coexpression

a)Diagrama de flujo



b)Explicación del proceso

La entrada de datos para el programa de coexpresión se realiza a través de la llamada al programa por línea de comandos (aunque se automatiza en Python de forma programática) con argumentos en la llamada, con la forma general:

```
python coexpression.py geneA1,geneA2,geneA3,...,geneAn
geneB1,geneB2,...,geneBn datafile correlation
```

Donde geneA*i* sea el nombre de cada gen de la primera famila de genes *A* y de igual forma geneB*i* sean los nombres de genes de la segunda familia *B*. Los genes deben separarse por comas (sin espacios) y ambas familias mediante un espacio.

El tercer y cuarto argumentos deben ser respectivamente el fichero con los datos a utilizar y el tipo de correlación a calcular (“pearson” o “spearman”).

Puede ser que un mismo gen se haya hibridado con distintas sondas (réplicas técnicas) en el microarray y por consiguiente encontraremos un dato por sonda. Los valores de las distintas réplicas son promediados. El promedio ha de ser por orden (promediamos la posición 1 de las diferentes sondas, luego la 2, etc.), de esta manera podremos establecer la correlación entre dos genes (sus listas de 48 datos) donde se emparejarán diferentes tejidos y estados transcripcionales para calcular la correlación.

Una vez que tenemos las listas de datos de todos los genes, llega la hora de establecer las correlaciones. Nos interesa calcular una correlación de cada gen de la primera lista de genes (familia A) con cada gen de la segunda lista de genes (familia B), lo que conseguimos fácilmente con dos bucles anidados. Para cada combinación realizamos una llamada a la función de R “cor” a través de la interfaz rpy2:

```
R.r.cor(v1,v2)[0]
```

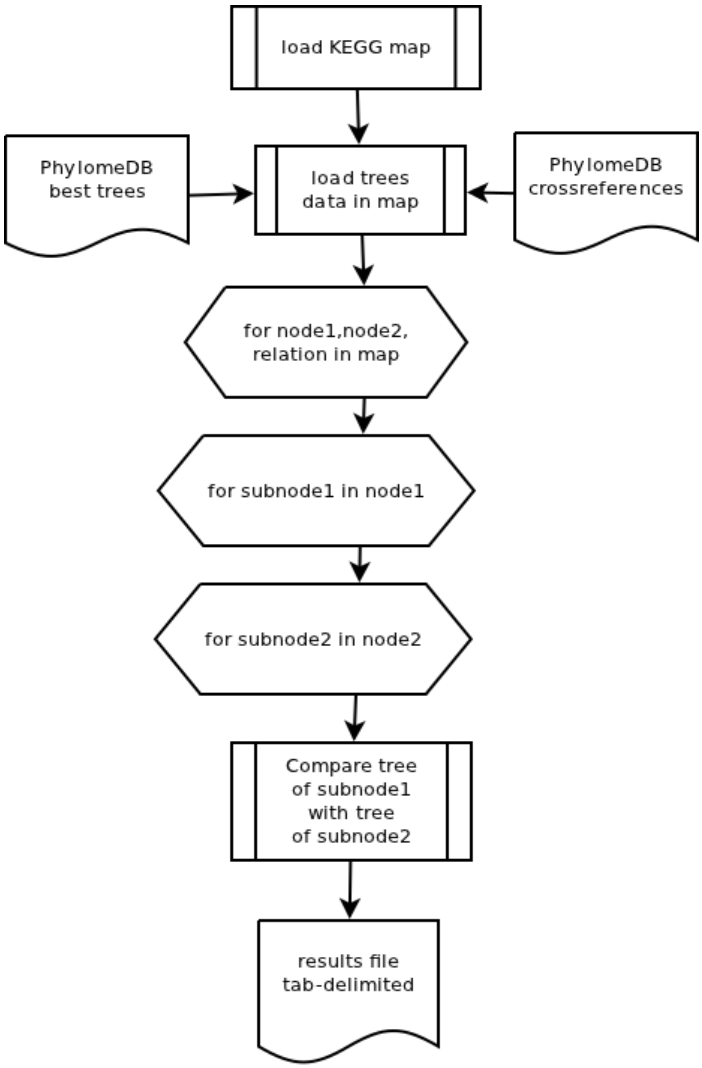
Donde v1 y v2 son los dos vectores de datos para los que hay que calcular la correlación. Como esta función devuelve un vector R (una lista de Python a través de rpy2), debemos acceder al primer (y único) elemento de la lista mediante la sintaxis [0].

Con todas las correlaciones calculadas se tabulan los resultados en un fichero de texto delimitado por tabulaciones en forma de matriz, donde una lista de genes ocupa la primera línea y la otra lista de genes la primera columna. Estos ficheros de texto serán los datos de entrada para el programa auxiliar que genera la representación gráfica de los datos (véase sección al final del capítulo).

El código fuente de este programa puede consultarse en el apéndice III sección a.

5.4. Programa coevolution

a)Diagrama de flujo



b)Explicación del proceso

Para calcular los resultados de coevolución, se ha enlazado los datos de KEGG con un algoritmo de recuperación de datos de PhylomeDB [3]. Posteriormente se utilizan dichos datos para obtener el árbol filogenético por gen (que se corresponde con subelemento KEGG), y se combinan en permutaciones para compararlos uno a uno mediante TreeKO [9].

El único parámetro que requiere este módulo es el identificador KEGG para el mapa que queramos procesar (en el caso de las MAPK es el “hsa04010”, que como cualquier

identificador de interés puede ser consultado en línea en la base de datos KEGG). A partir de este se recogen los datos de KEGG (opcionalmente pueden recogerse de un fichero local o guardarse para posteriores estudios).

A partir de los elementos KEGG (familias de genes) se extraen los subelementos (genes parálogos), y de estos referencias cruzadas a diferentes bases de datos. Son estas referencias cruzadas las que utiliza el programa para extraer los identificadores de PhylomeDB para esos genes. El escenario ideal sería aquel en que no existiera duplicidad en los identificadores, pero esas duplicidades existen con bastante frecuencia. En concreto, en este proceso se han encontrado varios identificadores de gen de PhylomeDB para un subnodo (o gen parálogo) KEGG. A partir de cada identificador de gen de PhylomeDB se busca un árbol filogenético, así que la duplicidad se transmite. No obstante en la práctica cada nodo de KEGG acaba teniendo un sólo árbol con todos (o la mayoría) de sus genes representados en él, pero el proceso no está exento de problemas.

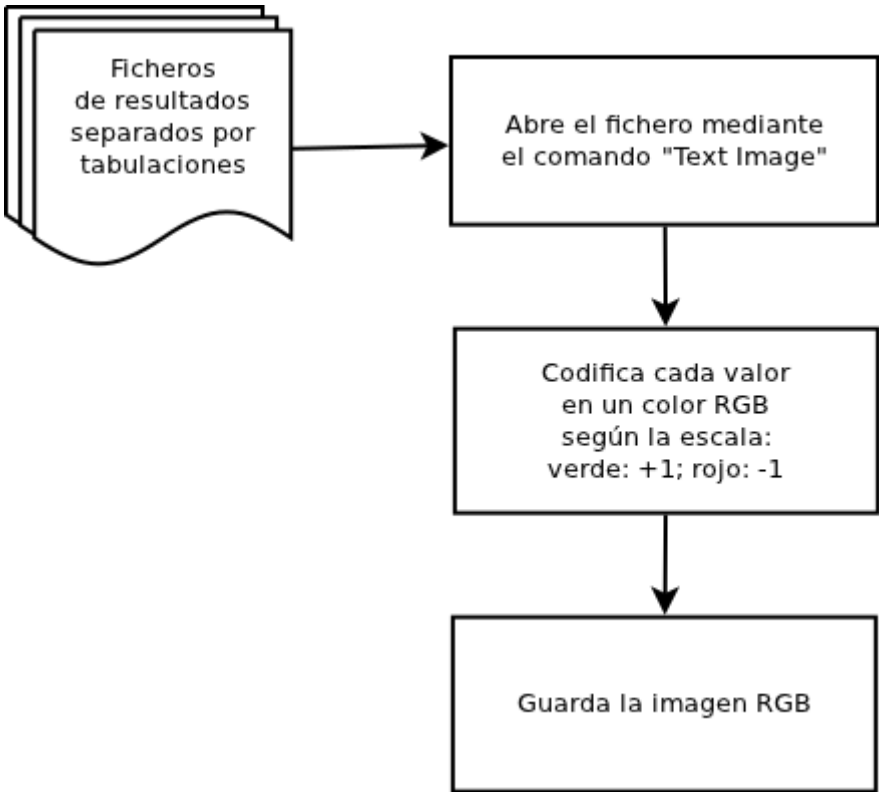
Por tanto para cada subnodo KEGG (o gen parálogo) debemos obtener un árbol filogenético con la información evolutiva de dicho gen en referencia a otros. Este árbol es lo que comparamos a otros árboles de genes relacionados para establecer según la semejanza de topologías si ha podido existir una evolución conjunta o coevolución. El cálculo lo realizamos a través del programa TreeKO [9] y obtenemos una distancia que indica semejanza en sus topologías. La distancia estará en un rango entre 0 y 1, y la mayor semejanza sería para una distancia de 0.

De la misma manera que en el programa coexpression se tabulan los resultados en un fichero por cada relación estudiada. Posteriormente se representan en matrices gráficas mediante un código de falso color (0 → verde; 1 → negro). Los detalles se pueden consultar en la sección 5.5 en que se explica la representación gráfica.

El código fuente del programa se puede consultar en el apéndice III sección d.

5.5. Representación de los datos

a)Diagrama de flujo



b)Explicación del proceso

Mediante el programa de análisis y procesamiento de imagen ImageJ [14], se ha programado una sencilla macro para transformar los resultados de correlación y distancia en matrices de falso color. Se han hecho dos versiones ligeramente distintas para realizar las matrices de coexpresión o las de coevolución.

Se ha elegido el rango rojo/verde para una lectura más fácil para quien esté familiarizado con las representaciones de microarrays, donde el rojo (RGB 0xFF0000 en notación hexadecimal) indica una correlación de -1 y el verde (RGB 0x00FF00) indica una correlación de +1. La ausencia total de correlación 0, está representada por el negro (RGB 0x000000).

En el caso de la representación de la distancia de los árboles evolutivos de las moléculas, se utiliza únicamente el rango [0,+1] en verde/negro, sólo que el 0 corresponde al verde y el +1 al negro, puesto que el concepto es inverso (una distancia de 0 indica identidad total entre ambos árboles y de +1 una distancia máxima y una identidad nula).

Se ha procurado seguir la estrategia más sencilla. ImageJ permite abrir documentos de texto plano como si fueran imágenes, de manera que busca valores tabulados y los convierte en una matriz de valores de imagen (con un máximo de precisión de 32-bit). Si existen valores NaN (not-a-number o no numéricos), estos son representados como valor 0, como por ejemplo las etiquetas de texto de filas y columnas en una tabla. Así, si abrimos de esta manera un fichero con datos tabulados separados mediante tabulaciones, obtendremos una imagen con una anchura igual al número de columnas y una altura igual al número de filas (en ambos casos incluyendo las etiquetas de texto). Si en la primera fila y columna tenemos las etiquetas de texto, encontraremos en la imagen la primera fila y columna todos los píxeles con un valor de 0.

6. RESULTADOS

En muchas de las parejas de genes comparadas no se ha podido aportar un resultado por falta de datos, y si existen datos, los cálculos reflejan una señal muy leve, y una significación escasa. A continuación se detallan los mejores resultados obtenidos considerando las correlaciones mayores o iguales a 0.40 para coexpresión y las distancias menores o iguales a 0.60 para coevolución.

6.1. Resultados de coexpresión

Para representar la correlación de Pearson se usa el símbolo r_P*, y la de Spearman* r_S*.*

- RASGRF1, CACNB4 r_P= 0.46
- RASGRF1, CACNB2 r_S= 0.43
- RASGRP3, CACNA1C r_P= 0.44
- IKBKB, NFKB1 r_P= 0.57
- IKBKB, RELA r_P= 0,44
- IKBKG, RELA r_P= 0,45r_S= 0.43
- IKBKG, RELB r_P= 0,46
- MAPK14, DUSP6 r_P= 0.46 r_S= 0.55
- MAPK11, DUSP3 r_P= 0.51
- MAPK11, DUSP8 r_P= 0.48
- MAPK13, DUSP5 r_P= 0.45
- MAPK12, DUSP9 r_P= 0.51
- MAPK1, DUSP6 r_P= 0.51 r_S= 0.42
- MAPK1, DUSP7 r_P= 0.51 r_S= 0.47
- MAPK3, DUSP7 r_P= 0.51 r_S= 0.58
- MAPK3, DUSP14 r_P= 0.43
- MAPK3, DUSP8 r_S= 0.51
- MAPK8, DUSP5 r_P= 0.40
- MAPK8, DUSP9 r_P= 0.49
- MAPK9, DUSP14 r_P= 0.40
- MAPK10, DUSP14 r_P= 0.49
- FGFR1, FGF7 r_S= 0.60
- FGFR3, FGF1 r_P= 0.59 r_S= 0.60
- FGFR2, FGF1 r_P= 0.59 r_S= 0.81
- FGFR2, FGF1 r_P= 0.47 r_S= 0.59
- FGFR2, FGF17 r_P= 0.41
- FGFR14, FGF3 r_P= 0.64
- FGFR14, FGF8 r_P= 0.48
- FGFR14, FGF16 r_P= 0.48
- MAPK9, HSPA2 r_P= 0.49
- MAPK10, HSPA2 r_P= 0.40
- IL1R2, IL1B r_P= 0.76 r_S= 0.55
- RPS6KA4, MAPK12 r_P= 0.41
- MAPK1, RPS6KA1 r_P= 0.42
- MAPK3, RPS6KA2 r_P= 0.44
- MAPK8IP2, MAPK8 r_P= 0.43

- MAPK8IP2, MAPK9 r_S= 0.50
- MAPK8IP2, MAPK10 r_P= 0.41 r_S= 0.54
- MAPK8IP1, MAPK9 r_S= 0.51
- MAPK8IP1, MAPK10 r_P= 0.46 r_S= 0.54
- RAP1A, PRKACB r_P= 0.50
- PTPRR, MAPK11 r_S= 0.60
- PAK1, RAC2 r_S= 0.43
- PAK1, CDC42 r_P= 0.67 r_S= 0.59
- PAK2, RAC2 r_P= 0.72 r_S= 0.82
- PAK2, CDC42 r_P= 0.68 r_S= 0.77
- RASGRF1, NRAS r_P= 0.42
- RASGRP1, KRAS r_P= 0.51
- SOS1, RRAS2 r_S= 0.61
- SOS1, NRAS r_S= 0.47
- SOS2, NRAS r_P= 0.50
- TGFBR2, TGFB1 r_P= 0.49 r_S= 0.48

Puede observarse que en la mayoría de estos resultados el coeficiente de correlación de Pearson es mayor que el de Spearman.

6.2. Resultados de coevolución

Sólo se han calculado las siguientes distancias menores o iguales al umbral de 0.60 considerado.

- PDGFRB, PDGFB *distancia* = 0.33
- MAPK11, DUSP16 *distancia* = 0.60
- MAPK12, DUSP16 *distancia* = 0.60
- MKNK2, MAPK3 *distancia* = 0.57

Los resultados calculados con la estrategia de coevolución son menos significativos de los encontrados con la coexpresión. Además no se ha encontrado ninguna pareja de genes para la que tanto el resultado de coexpresión y de coevolución sean significativos.

7. DISCUSIÓN

Uno de los resultados principales de este proyecto es el diseño de una serie de programas para responder a los objetivos. Si bien la solución diseñada da respuesta al problema definido, puede ser considerado como un prototipo para futuros desarrollos. Los cálculos realizados son susceptibles de mejoras y la suite informática diseñada se puede completar con nuevos módulos.

Python es un lenguaje de alto nivel que permite el desarrollo de programas de forma ágil. Al tratarse de un lenguaje multiparadigma permite el enfoque utilizado en este proyecto, que usa una estrategia mixta: programación funcional y orientada a objetos.

El rendimiento que podría verse perjudicado por la elección de un lenguaje de mayor nivel (como Python frente a C, por ejemplo) no ha sido tenido en cuenta como un limitante y tampoco ha resultado un problema, ya que los programas codificados están diseñados para ser ejecutados en pocas ocasiones, por lo que este es un problema menor. De hecho la elección de Python redunda en un menor tiempo de desarrollo y una depuración más rápida y precisa.

Otra razón importante de la elección de Python ha sido la necesidad del uso de ETE2 [10] para el módulo de coevolución, que es una toolkit escrita en Python, por lo que la comunicación con esta es directa a través de programas Python. Asimismo la comunicación con KEGG se realiza a través de SOAP mediante las definiciones de los datos en fichero WSDL.

El paquete estadístico R fue escogido para la realización de las correlaciones entre datos de expresión, por lo que se hizo presente la necesidad de comunicación de datos y ejecución entre Python y R. Se estudiaron diferentes soluciones, entre las que se escogió el uso de una interfaz entre Python y R denominada rpy2 que permite el acceso a las funciones y estructuras de datos de R a través de Python. La utilización de una interfaz simplifica las estrategias de cálculo de modo que pueden mezclarse datos y funciones tanto de Python como de R, por lo que el resultado es un código exclusivamente escrito en Python, claro y fácil de mantener.

El enfoque utilizado en este proyecto es el de utilizar diferentes fuentes de datos para integrar la información contenida en ellas. Se utiliza una estrategia de minería de datos considerando una fuente principal: la base de datos KEGG y dos fuentes que aportan los datos de relaciones entre genes. El resultado es satisfactorio aunque subóptimo. Se

considera que los resultados mejorarían al incrementar la calidad de los datos y/o de los algoritmos utilizados para su procesamiento.

Los datos utilizados son escasos o bien contienen una información limitada. Este hecho se pone de manifiesto al encontrar datos nulos o con una señal tenue en muchos casos, que lleva a que existan multitud de relaciones sin resultado en absoluto o con una significación muy leve. Para la obtención de mejores resultados convendría utilizar datos de mayor calidad. En general en los resultados se observa que la coexpresión aporta poca señal y la coevolución muy poca señal para descubrir relaciones entre parejas de genes.

Existen grandes diferencias entre los resultados obtenidos mediante los datos de expresión y los de evolución. De realizar una optimización de los algoritmos deberían considerarse estrategias diferentes para cada uno de los dos enfoques propuestos. Se sugiere utilizar más datos y/o de mayor calidad para mejorar la coexpresión calculada: mayor número de microarrays para incrementar la relación señal/ruido y búsqueda de sinónimos de genes o referencias cruzadas para disminuir los datos ausentes. Por otro lado los algoritmos en búsqueda y procesamiento de datos para la coevolución podrían mejorarse utilizando la secuencia de aminoácidos para realizar una búsqueda a través de BLAST [15], opción que ofrece la página web de PhylomeDB, en los casos en que ninguna de las referencias cruzadas almacenadas en KEGG sirva para recoger datos. Sin embargo como esta opción no está integrada en la API para implementarla habría que utilizar una librería que automatizara la navegación de páginas web, como mechanize , que al estar escrita en Python sería de más fácil implementación (http://wwwsearch.sourceforge.net/mechanize/).

Dada la mayor complejidad conceptual y de método de la evolución de secuencias, los algoritmos utilizados para elucidar la coevolución son más sofisticados y probablemente menos fiables. En este sentido las posibles mejoras en los algoritmos que realizan las comparaciones serán fundamentalmente referidas a los de coevolución.

Este estudio se ha realizado para la vía metabólica de señalización de las MAPK en humanos, y establece que no es fácil encontrar relaciones basadas en el análisis de coexpresión y coevolución para este caso concreto. Esta dificultad puede ser debida a la existencia de gran cantidad de parálogos para cada nodo de la vía complicando la búsqueda de relaciones, es decir, probablemente se trate de un caso difícil y los escasos resultados pueden ser debidos a ello. Si se repitiera este estudio para otro caso distinto a la vía de las MAPK, probablemente se obtendrían mejores resultados, en concreto es posible que ocurra de estudiar las relaciones entre genes implicados en un complejo macromolécula, como por

ejemplo el proteosoma, donde existan mayores presiones funcionales y evolutivas entre unos y otros componentes del sistema.

A pesar de las dificultades y los escasos resultados, existen casos positivos en que se ha encontrado una coexpresión y coevolución positivas, lo cual demuestra que la estrategia seguida en el estudio puede ser eficaz.

8. CONCLUSIONES Y TRABAJO FUTURO

8.1. Conclusiones

- El desarrollo de una **plataforma** para el **estudio** de las **relaciones** entre proteínas contenidas en vías metabólicas y de señalización humanas, se considera un éxito en sí mismo. En este sentido, este trabajo sirve para establecer un punto de partida para otros estudios en la misma dirección.
- Hemos comprobado que utilizando cálculos de **coexpresión** y **coevolución** se pueden establecer **preferencias de relación** entre proteínas concretas de dos familias de parálogos que interactúan. Los métodos y series de datos utilizados han dado resultados satisfactorios, si bien mejorables.
- Las **correlaciones** conseguidas en ocasiones no son óptimas, pero indican una **tendencia**. Se ha aplicado un tratamiento estadístico lo más robusto posible para obtener los resultados. Entendemos que es posible mejorar tanto el tipo de datos transcriptómicos como el conocimiento del árbol evolutivo de las proteínas analizados y que esto llevará a unos resultados más optimizados.
- El reconocimiento de **coevolución** entre pares de genes es un concepto complejo y requiere de algoritmos sofisticados que manejen gran cantidad de combinaciones posibles de los cambios evolutivos. En este trabajo hemos mostrado una aproximación que puede dar resultados satisfactorios.

8.2. Trabajo Futuro

Para completar este trabajo convendría afrontar los problemas de falta de datos que llevan a una carencia en los resultados. Por una parte esto podría ser abordado mejorando el procesamiento de los datos, por otra mejorando su calidad o aumentando su cantidad. En concreto se sugiere que se consideren más fuentes de datos que puedan aportar más información al problema planteado, por ejemplo otra serie de microarrays para los cálculos de coexpresión u otras estrategias de cálculo.

Asimismo se podría mejorar la integración de los datos, de modo que como resultado se aporte una medida que tengan en cuenta las diferentes fuentes de datos en vez de un resultado por cada fuente de datos.

CONCLUSIONES Y TRABAJO FUTURO

El planteamiento de este proyecto es la extracción de información de una vía concreta, pero podría ampliarse para procesar otras. De obtener unos resultados lo suficientemente significativos, se debería buscar algún modo de integrar esta información y presentarla visualmente del modo más adecuado para permitir su consulta en Internet.

Para adecuar la visualización de los resultados, se podría hacer uso de un conjunto de herramientas para KEGG denominado KEGG Mapper [4]. También podrían representarse en iCanPlot [16].

9. REFERENCIAS Y ENLACES

[1] S. Goto, H. Bono, H. Ogata, W. Fujibuchi, T. Nishioka, K. Sato, M. Kanehisa, "Organizing and computing metabolic pathway data in terms of binary relations", *Pacific Symposium on Biocomputing*, 175-86, 1997.

[2] J. Huerta-Cepas, H. Dopazo, J. Dopazo, T. Gabaldón, "The human phylome", *Genome Biology*, vol. 8, issue 6, R109, 2007.

[3] J. Huerta-Cepas, A. Bueno, J. Dopazo, T. Gabaldon, "PhylomeDB: a database for genome-widecollections of gene phylogenies", *Nucleic Acids Research*, vol. 36, Database issue, pp.D491-D496, 2008.

[4] M. Kanehisa, S. Goto, Y. Sato, M. Furumichi, M. Tanabe, "KEGG for integration and interpretation oflarge-scale molecular data sets", *Nucleic Acids Research*, vol. 40, Database issue, pp. D109-D114, 2012.

[5] P. Khatri, M. Sirota, A. J. Butte, "Ten Years of Pathway Analysis: Current Approaches and Outstanding Challenges", *PLOS Computational Biology*, vol. 8, issue 2, pp. 1-10, 2012.

[6] J. Gollub, C. A. Ball, G. Binkley, J. Demeter, D. B. Finkelstein, J. M. Hebert, T. Hernandez-Boussard, H. Jin, M. Kaloper, J. C.Matese, M. Schroeder, P. O. Brown, D. Botstein, G. Sherlock, "The Stanford Microarray Database: data access and quality assessment tools", *Nucleic Acids Research*, vol. 31, No. 1, pp. 94-96, 2003.

[7] C. Prieto, A. Risueño, C. Fontanillo, J. De Las Rivas, "Human Gene Coexpression Landscape: Confident Network Derived from Tissue Transcriptomic Profiles", *PLOS ONE*, vol. 3, issue 12, pp. 1-14, 2008.

[8] J. Huerta-Cepas, S. Capella-Gutierrez, L. P. Pryszcz, I. Denisov, ́D. Kormes, M. Marcet-Houben, T. Gabaldon, "PhylomeDB v3.0: an expanding repository of genome-wide collections of trees, alignments and phylogeny-based orthology and paralogy predictions", *Nucleic Acids Research*, vol. 39, Database issue, pp. 556-560, 2011.

[9] M. Marcet-Houben, T. Gabaldon, "TreeKO: a duplication-aware algorithm for the comparison of phylogenetic trees", *Nucleic Acids Research*, vol. 39, No. 10, e66, pp. 1-10, 2011.

[10] J. Huerta-Cepas, J. Dopazo, T. Gabaldón, "ETE: a python Environment for Tree Exploration", *BCM Bioinformatics*, 11:24, 2010.

[11] R Development Core Team, "R: A Language and Environment for Statistical Computing", *R Foundation for Statistical Computing*, ISBN 3-900051-07-0, 2010.

[12] *rpy2: a Python interface to the R Programming Language*, http://rpy.sourceforge.net

[13] G. Van Rossum, "Scripting Languages: Automating the Web - Scripting the Web with Python", *World Wide Web Journal*, vol. 2, issue 2, 1997.

[14] M. D. Abramoff, P. J. Magalhaes, S. J. Ram, "Image Processing with ImageJ", *Biophotonics International*, vol. 11, issue 7, pp. 36-42, 2004.

[15] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, D. J. Lipman, "Basic local alignment search tool", *Journal of Molecular Biology*, , 1990.

[16] A. U. Sinha, S. A. Armstrong, "iCanPlot: Visual Exploration of High-Throughput Omics Data Using Interactive Canvas Plotting", *PLOS ONE*, vol.7, issue 2, e31690, pp. 1-4, 2012.

10. APÉNDICES

10.1. Apéndice I. GLOSARIO Y ACRÓNIMOS

- **API**: (del inglés application programming interface) interfaz de programación de aplicaciones, es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
- **BBDD**: base(-s) de datos.
- **CIC**: Centro de Investigación del Cáncer de la Universidad de Salmanca)
- **CRG**: El Centro de Regulación Genómica (CRG) es un innovador centro de investigación básica creado en diciembre de 2000 por iniciativa del Departamento de Universidades, Investigación y Sociedad de la Información (DURSI) de la Generalitat de Catalunya.
- **Elemento**, supergen o familia de genes: abstracción del conjunto de genes. Es el elemento legible en la vía de KEGG dentro de un rectángulo. En realidad, estos elementos son conjuntos de genes (o de subelementos).
- **KEGG**: Kyoto Encyclopedia of Genes and Genomes.
- **MAPK**: The mitogen-activated protein kinase (MAPK) cascade is a highly conserved module that is involved in various cellular functions, including cell proliferation, differentiation and migration. (http://www.genome.jp/kegg/pathway/hsa/hsa04010.html)
- **SOAP**: (siglas de Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. (Fuente: Wikipedia)
- **Subelemento,** gen o parálogo: unidad mínima de información secuencial (proteínica), dentro de la representación de la vía (como subelemento) está por debajo del conjunto o elemento también llamado supergen.
- **WSDL**: son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios Web (Fuente: Wikipedia)

10.2. Apéndice II. TABLAS DE RESULTADOS COMPLETAS

A continuación se presentan las tablas de resultados completas. En la sección resultados se han incluido sólo los resultados que se han considerado significativos.

a)Resultados de coexpresión

Estas tablas contienen la correlación de pearson o spearman (indicado en cada caso), y de fondo se muestra esta misma información con un código de color, donde el 0 se representa en negro, el -1 en rojo y el +1 en verde (mediante una transformación del dato a una suma RGB), como se representa en la escala facilitada a continuación. Para ello se ha programado un sencillo script en ImageJ [14], un programa de análisis y procesamiento de imagen del National Institute of Health. El código de este script puede consultarse en el apéndice III sección f.

Los datos ausentes se han codificado con la palabra “None”, se corresponden con los genes que no se han encontrado en los ficheros de expresión de los microarrays.

Los resultados obtenidos pueden indicar una tendencia de relación de los genes, observable en las diferencias existentes entre las correlaciones en una misma línea o columna. Se observan algunos genes sin datos, lo cual podría mejorar al manejar un número mayor de sinónimos o la existencia de referencias cruzadas directas entre la base de datos KEGG y los microarrays de Affymetrix.

CACNG3, CACNG2, CACNG5, CACNG4, CACNA2D3, CACNG8, CACNG7, CACNG6, CACNA1A, CACNA1B, CACNA1C, CACNA1D, CACNA1E, CACNA1F, CACNA1S, CACNA2D1, CACNB1, CACNB2, CACNB3, CACNB4, CACNG1, CACNA1I, CACNA1H, CACNA1G, CACNA2D2, CACNA2D4 * RASGRF1, RASGRF2

Pearson

genes	RASGRF1	RASGRF2
CACNG3	0.24	None
CACNG2	0.26	None
CACNG5	0.28	None
CACNG4	0.19	None
CACNA2D3	-0.04	None
CACNG8	None	None
CACNG7	None	None
CACNG6	None	None
CACNA1A	0.15	None
CACNA1B	-0.03	None
CACNA1C	0.39	None
CACNA1D	0.35	None
CACNA1E	0.12	None
CACNA1F	0.15	None
CACNA1S	0.12	None
CACNA2D1	0.24	None
CACNB1	-0.02	None
CACNB2	0.15	None
CACNB3	-0.04	None
CACNB4	0.46	None
CACNG1	0.27	None
CACNA1I	0.26	None
CACNA1H	-0.27	None
CACNA1G	0.32	None
CACNA2D2	0.03	None
CACNA2D4	None	None

Spearman

genes	RASGRF1	RASGRF2
CACNG3	0.34	None
CACNG2	0.14	None
CACNG5	0.11	None
CACNG4	-0.30	None
CACNA2D3	0.17	None
CACNG8	None	None
CACNG7	None	None
CACNG6	None	None
CACNA1A	0.22	None
CACNA1B	-0.04	None
CACNA1C	0.32	None
CACNA1D	0.11	None
CACNA1E	0.12	None
CACNA1F	0.10	None
CACNA1S	-0.08	None
CACNA2D1	0.14	None
CACNB1	0.12	None
CACNB2	0.43	None
CACNB3	0.08	None
CACNB4	-0.05	None
CACNG1	-0.03	None
CACNA1I	0.39	None
CACNA1H	-0.09	None
CACNA1G	0.23	None
CACNA2D2	-0.11	None
CACNA2D4	None	None

CACNG3, CACNG2, CACNG5, CACNG4, CACNA2D3, CACNG8, CACNG7, CACNG6, CACNA1A, CACNA1B, CACNA1C, CACNA1D, CACNA1E, CACNA1F, CACNA1S, CACNA2D1, CACNB1, CACNB2, CACNB3, CACNB4, CACNG1, CACNA1I, CACNA1H, CACNA1G, CACNA2D2, CACNA2D4 * RASGRP1, RASGRP2, RASGRP4, RASGRP3

Pearson

genes	RASGRP1	RASGRP2	RASGRP4	RASGRP3
CACNG3	-0.07	-0.18	None	0.05
CACNG2	-0.19	0.18	None	0.00
CACNG5	-0.39	-0.05	None	0.21
CACNG4	-0.40	-0.06	None	0.07
CACNA2D3	0.35	0.14	None	-0.06
CACNG8	None	None	None	None
CACNG7	None	None	None	None
CACNG6	None	None	None	None
CACNA1A	-0.10	0.14	None	0.07
CACNA1B	-0.38	0.05	None	0.10
CACNA1C	-0.39	0.06	None	0.44
CACNA1D	-0.42	-0.25	None	0.04
CACNA1E	-0.26	0.17	None	0.18
CACNA1F	-0.23	0.09	None	-0.07
CACNA1S	-0.09	0.06	None	0.05
CACNA2D1	-0.32	-0.11	None	0.08
CACNB1	-0.11	-0.15	None	-0.13
CACNB2	-0.30	-0.23	None	-0.03
CACNB3	-0.15	-0.31	None	-0.19
CACNB4	-0.23	0.03	None	0.14
CACNG1	-0.26	0.01	None	-0.02
CACNA1I	-0.19	0.17	None	0.30
CACNA1H	-0.36	-0.06	None	-0.23
CACNA1G	-0.32	0.08	None	0.18
CACNA2D2	-0.26	-0.19	None	-0.21
CACNA2D4	None	None	None	None

Spearman

genes	RASGRP1	RASGRP2	RASGRP4	RASGRP3
CACNG3	0.03	-0.14	None	-0.09
CACNG2	-0.11	-0.02	None	-0.18
CACNG5	-0.22	-0.20	None	-0.10
CACNG4	-0.16	0.18	None	0.08
CACNA2D3	0.22	0.02	None	0.03
CACNG8	None	None	None	None
CACNG7	None	None	None	None
CACNG6	None	None	None	None
CACNA1A	-0.17	0.02	None	0.14
CACNA1B	-0.19	-0.16	None	0.04
CACNA1C	-0.13	0.09	None	0.04
CACNA1D	-0.33	-0.45	None	0.15
CACNA1E	-0.21	0.17	None	0.01
CACNA1F	-0.24	-0.12	None	-0.11
CACNA1S	-0.22	0.13	None	0.11
CACNA2D1	-0.14	0.02	None	-0.21
CACNB1	-0.19	-0.06	None	-0.13
CACNB2	-0.17	-0.26	None	-0.12
CACNB3	-0.10	-0.34	None	-0.11
CACNB4	0.03	-0.17	None	0.01
CACNG1	-0.24	-0.10	None	0.08
CACNA1I	-0.18	0.03	None	0.01
CACNA1H	-0.24	-0.03	None	-0.17
CACNA1G	-0.29	0.03	None	-0.11
CACNA2D2	-0.10	-0.01	None	-0.01
CACNA2D4	None	None	None	None

CHUK, IKBKB, IKBKG * NFKB1, NFKB2, RELA, RELB

Pearson

genes	CHUK	IKBKB	IKBKG
NFKB1	0.13	0.57	0.21
NFKB2	-0.13	0.23	0.25
RELA	-0.12	0.44	0.45
RELB	-0.13	0.29	0.46

Spearman

genes	CHUK	IKBKB	IKBKG
NFKB1	0.15	0.31	0.17
NFKB2	0.25	0.31	0.23
RELA	-0.37	0.03	0.43
RELB	-0.00	0.12	0.32

DUSP14, DUSP10, DUSP1, DUSP2, DUSP3, DUSP4, DUSP5, DUSP6, DUSP7, DUSP8, DUSP9, DUSP22, DUSP16 * MAPK14, MAPK11, MAPK13, MAPK12

Pearson

genes	MAPK14	MAPK11	MAPK13	MAPK12
DUSP14	-0.29	0.21	0.08	-0.02
DUSP10	-0.14	0.24	0.11	0.21
DUSP1	0.25	-0.24	-0.07	-0.06
DUSP2	0.21	-0.03	-0.14	-0.18
DUSP3	-0.26	0.51	-0.06	0.35
DUSP4	-0.21	0.37	0.21	0.02
DUSP5	-0.19	-0.19	0.47	-0.03
DUSP6	0.46	-0.30	-0.14	-0.25
DUSP7	0.08	0.21	-0.19	0.18
DUSP8	-0.17	0.48	-0.32	0.13
DUSP9	-0.22	0.33	0.18	0.51
DUSP22	-0.05	-0.06	0.09	-0.18
DUSP16	None	None	None	None

Spearman

genes	MAPK14	MAPK11	MAPK13	MAPK12
DUSP14	-0.30	-0.15	-0.11	-0.17
DUSP10	-0.16	0.10	0.02	0.11
DUSP1	0.17	-0.31	0.05	-0.07
DUSP2	-0.00	-0.08	-0.28	-0.21
DUSP3	-0.35	0.31	-0.18	0.22
DUSP4	-0.11	-0.10	0.08	-0.18
DUSP5	-0.07	-0.06	0.39	0.03
DUSP6	0.55	-0.38	-0.11	-0.14
DUSP7	-0.02	0.19	-0.09	0.15
DUSP8	-0.09	0.26	-0.22	0.11
DUSP9	-0.10	0.31	0.03	0.21
DUSP22	-0.05	-0.13	0.29	-0.08
DUSP16	None	None	None	None

DUSP14, DUSP10, DUSP1, DUSP2, DUSP3, DUSP4, DUSP5, DUSP6, DUSP7, DUSP8, DUSP9, DUSP22, DUSP16 * MAPK1, MAPK3

Pearson

genes	MAPK1	MAPK3
DUSP14	0.17	0.32
DUSP10	-0.19	-0.11
DUSP1	-0.01	0.17
DUSP2	0.11	0.03
DUSP3	0.02	0.22
DUSP4	-0.23	-0.08
DUSP5	-0.11	-0.05
DUSP6	0.51	0.05
DUSP7	0.51	0.51
DUSP8	0.11	0.28
DUSP9	-0.32	-0.04
DUSP22	0.36	0.19
DUSP16	None	None

Spearman

genes	MAPK1	MAPK3
DUSP14	0.03	0.43
DUSP10	-0.13	-0.47
DUSP1	-0.05	0.23
DUSP2	0.10	0.26
DUSP3	-0.11	0.33
DUSP4	-0.04	-0.10
DUSP5	-0.08	-0.19
DUSP6	0.42	0.14
DUSP7	0.47	0.58
DUSP8	0.38	0.51
DUSP9	0.07	-0.16
DUSP22	0.14	0.26
DUSP16	None	None

DUSP14, DUSP10, DUSP1, DUSP2, DUSP3, DUSP4, DUSP5, DUSP6, DUSP7, DUSP8, DUSP9, DUSP22, DUSP16 * MAPK8, MAPK9, MAPK10

Pearson

genes	MAPK8	MAPK9	MAPK10
DUSP14	-0.04	0.40	0.49
DUSP10	0.18	-0.12	-0.09
DUSP1	-0.22	-0.16	-0.25
DUSP2	-0.26	-0.03	-0.45
DUSP3	0.06	0.16	0.21
DUSP4	0.07	0.03	-0.07
DUSP5	0.40	-0.06	-0.31
DUSP6	-0.44	0.15	-0.32
DUSP7	-0.26	0.38	0.14
DUSP8	-0.32	0.16	0.22
DUSP9	0.49	-0.04	-0.01
DUSP22	-0.25	0.16	-0.21
DUSP16	None	None	None

Spearman

genes	MAPK8	MAPK9	MAPK10
DUSP14	-0.11	0.07	-0.05
DUSP10	0.05	-0.03	0.17
DUSP1	0.06	-0.20	-0.23
DUSP2	0.10	-0.02	-0.29
DUSP3	-0.07	0.26	0.19
DUSP4	-0.01	0.18	-0.20
DUSP5	0.16	-0.03	-0.35
DUSP6	-0.12	0.03	-0.40
DUSP7	-0.05	0.22	0.00
DUSP8	-0.03	0.16	0.33
DUSP9	0.23	-0.10	0.19
DUSP22	-0.15	-0.16	-0.31
DUSP16	None	None	None

FGF1, FGF2, FGF3, FGF4, FGF5, FGF6, FGF7, FGF8, FGF9, FGF10, FGF11, FGF12, FGF13, FGF14, FGF20, FGF21, FGF22, FGF23, FGF18, FGF17, FGF16, FGF19 * FGFR1, FGFR3, FGFR2, FGFR4

Pearson

genes	FGFR1	FGFR3	FGFR2	FGFR4
FGF1	-0.18	0.59	0.59	-0.16
FGF2	0.18	0.02	0.47	-0.34
FGF3	-0.02	0.10	0.06	0.64
FGF4	-0.03	-0.30	0.08	0.20
FGF5	0.11	-0.05	0.25	-0.10
FGF6	-0.15	0.26	0.16	0.27
FGF7	0.34	-0.23	0.26	-0.24
FGF8	0.08	0.10	0.06	0.48
FGF9	0.06	0.19	0.32	-0.02
FGF10	None	None	None	None
FGF11	None	None	None	None
FGF12	-0.33	0.18	0.33	-0.21
FGF13	-0.01	0.33	0.29	-0.29
FGF14	-0.18	0.24	0.28	-0.03
FGF20	0.08	-0.04	-0.14	0.23
FGF21	-0.08	0.22	0.27	0.15
FGF22	0.28	-0.11	-0.22	0.28
FGF23	0.10	-0.18	0.15	-0.04
FGF18	0.19	0.37	0.18	0.38
FGF17	-0.11	0.13	0.41	-0.04
FGF16	0.03	-0.03	-0.05	0.48
FGF19	None	None	None	None

Spearman

genes	FGFR1	FGFR3	FGFR2	FGFR4
FGF1	-0.24	0.60	0.81	-0.14
FGF2	0.20	0.18	0.59	-0.31
FGF3	0.15	0.07	-0.21	0.29
FGF4	0.23	-0.26	-0.19	-0.03
FGF5	0.41	0.03	0.02	0.06
FGF6	0.04	0.15	-0.10	0.04
FGF7	0.60	0.09	-0.02	-0.10
FGF8	0.22	-0.07	-0.02	0.10
FGF9	-0.03	0.08	0.12	0.04
FGF10	None	None	None	None
FGF11	None	None	None	None
FGF12	0.10	-0.05	0.15	-0.19
FGF13	-0.07	0.32	0.36	-0.20
FGF14	0.03	0.14	0.05	0.27
FGF20	0.18	0.13	-0.14	-0.08
FGF21	0.34	0.01	-0.04	0.29
FGF22	0.15	-0.16	-0.03	0.14
FGF23	-0.08	-0.11	-0.07	0.12
FGF18	0.37	0.00	0.09	0.05
FGF17	0.15	-0.25	0.09	0.00
FGF16	0.09	0.13	-0.16	0.03
FGF19	None	None	None	None

HSPA1A, HSPA1B, HSPA1L, HSPA2, HSPA6, HSPA8 * MAPK8, MAPK9, MAPK10

Pearson

genes	MAPK8	MAPK9	MAPK10
HSPA1A	-0.53	-0.06	-0.13
HSPA1B	-0.41	-0.05	0.08
HSPA1L	0.23	0.08	-0.15
HSPA2	-0.02	0.49	0.40
HSPA6	-0.31	-0.13	-0.44
HSPA8	-0.33	0.23	0.08

Spearman

genes	MAPK8	MAPK9	MAPK10
HSPA1A	-0.22	-0.01	-0.25
HSPA1B	-0.24	0.16	-0.12
HSPA1L	0.09	0.01	-0.06
HSPA2	-0.08	0.34	0.27
HSPA6	-0.09	-0.08	-0.32
HSPA8	-0.19	0.26	0.01

IL1A, IL1B * IL1R1, IL1R2

Pearson

genes	IL1R1	IL1R2
IL1A	0.03	0.06
IL1B	-0.38	0.76

Spearman

genes	IL1R1	IL1R2
IL1A	-0.03	-0.08
IL1B	-0.14	0.55

MAPK14, MAPK11, MAPK13, MAPK12 * MAPKAPK3, MAPKAPK2

Pearson

genes	MAPKAPK3	MAPKAPK2
MAPK14	0.34	0.03
MAPK11	0.01	-0.02
MAPK13	-0.10	0.20
MAPK12	0.24	0.11

Spearman

genes	MAPKAPK3	MAPKAPK2
MAPK14	0.12	-0.04
MAPK11	0.01	-0.06
MAPK13	-0.20	0.08
MAPK12	0.37	0.28

MAPK14, MAPK11, MAPK13, MAPK12 * RPS6KA4, RPS6KA5

Pearson

genes	RPS6KA4	RPS6KA5
MAPK14	0.05	0.21
MAPK11	0.17	-0.38
MAPK13	0.02	-0.27
MAPK12	0.41	-0.14

Spearman

genes	RPS6KA4	RPS6KA5
MAPK14	0.26	0.31
MAPK11	0.09	-0.09
MAPK13	-0.14	-0.17
MAPK12	0.10	-0.05

MAPK1, MAPK3 * MKNK2, MKNK1

Pearson

genes	MKNK2	MKNK1
MAPK1	0.09	0.38
MAPK3	0.02	-0.20

Spearman

genes	MKNK2	MKNK1
MAPK1	-0.08	-0.06
MAPK3	0.05	-0.53

MAPK1, MAPK3 * PLA2G4B, PLA2G4E, PLA2G2D, PLA2G2E, PLA2G2C, PLA2G3, PLA2G1B, PLA2G2A, PLA2G4A, PLA2G5, PLA2G2F, PLA2G12A, PLA2G6, PLA2G10, PLA2G12B, JMJD7-PLA2G4B

Pearson

genes	MAPK1	MAPK3
PLA2G4B	-0.23	0.11
PLA2G4E	None	None
PLA2G2D	-0.37	-0.09
PLA2G2E	-0.09	0.01
PLA2G2C	None	None
PLA2G3	-0.32	-0.25
PLA2G1B	-0.14	-0.29
PLA2G2A	-0.28	-0.03
PLA2G4A	-0.16	-0.17
PLA2G5	-0.04	0.17
PLA2G2F	-0.18	0.12
PLA2G12A	-0.17	-0.21
PLA2G6	-0.05	0.34
PLA2G10	-0.05	-0.05
PLA2G12B	None	None
JMJD7-PLA2G4B	None	None

Spearman

genes	MAPK1	MAPK3
PLA2G4B	-0.25	-0.20
PLA2G4E	None	None
PLA2G2D	-0.11	-0.27
PLA2G2E	-0.07	-0.17
PLA2G2C	None	None
PLA2G3	0.08	-0.31
PLA2G1B	-0.13	-0.14
PLA2G2A	-0.07	-0.21
PLA2G4A	-0.15	-0.09
PLA2G5	-0.10	0.01
PLA2G2F	-0.09	-0.15
PLA2G12A	-0.08	-0.08
PLA2G6	-0.09	-0.04
PLA2G10	-0.04	-0.10
PLA2G12B	None	None
JMJD7-PLA2G4B	None	None

MAPK1, MAPK3 * RPS6KA6, RPS6KA1, RPS6KA2, RPS6KA3

Pearson

genes	MAPK1	MAPK3
RPS6KA6	-0.30	-0.14
RPS6KA1	0.38	0.07
RPS6KA2	-0.03	0.23
RPS6KA3	0.16	-0.34

Spearman

genes	MAPK1	MAPK3
RPS6KA6	-0.18	-0.46
RPS6KA1	0.42	0.01
RPS6KA2	-0.02	0.44
RPS6KA3	-0.17	-0.38

MAPK8IP2, MAPK8IP1 * MAPK8, MAPK9, MAPK10

Pearson

genes	MAPK8IP2	MAPK8IP1
MAPK8	0.43	0.09
MAPK9	0.37	0.33
MAPK10	0.41	0.46

Spearman

genes	MAPK8IP2	MAPK8IP1
MAPK8	0.07	-0.01
MAPK9	0.50	0.51
MAPK10	0.54	0.54

MAPK8, MAPK9, MAPK10 * CRK, CRKL

Pearson

genes	CRK	CRKL
MAPK8	-0.35	-0.35
MAPK9	0.19	0.06
MAPK10	0.12	-0.18

Spearman

genes	CRK	CRKL
MAPK8	-0.39	0.11
MAPK9	0.12	-0.15
MAPK10	-0.02	0.04

NTF3, NTF4 * NTRK1, NTRK2

Pearson

genes	NTRK1	NTRK2
NTF3	0.32	-0.28
NTF4	None	None

Spearman

genes	NTRK1	NTRK2
NTF3	0.13	-0.06
NTF4	None	None

PAK1, PAK2 * MAP3K2, MAP3K3

Pearson

genes	MAP3K2	MAP3K3
PAK1	0.27	0.13
PAK2	0.28	0.14

Spearman

genes	MAP3K2	MAP3K3
PAK1	0.10	0.06
PAK2	0.17	-0.08

PDGFA, PDGFB * PDGFRA, PDGFRB

Pearson

genes	PDGFRA	PDGFRB
PDGFA	0.16	0.14
PDGFB	0.15	0.33

Spearman

genes	PDGFRA	PDGFRB
PDGFA	0.23	0.16
PDGFB	0.36	0.32

PRKACA, PRKACB, PRKACG, PRKX * RAP1A, RAP1B

Pearson

genes	RAP1A	RAP1B
PRKACA	-0.11	-0.13
PRKACB	0.50	0.34
PRKACG	-0.44	-0.57
PRKX	0.32	0.31

Spearman

genes	RAP1A	RAP1B
PRKACA	-0.32	-0.26
PRKACB	0.27	0.18
PRKACG	-0.25	-0.14
PRKX	0.37	0.23

PRKCA, PRKCB, PRKCG * RRAS2, MRAS, HRAS, KRAS, NRAS, RRAS

Pearson

genes	PRKCA	PRKCB	PRKCG
RRAS2	-0.05	None	-0.37
MRAS	0.33	None	0.39
HRAS	0.19	None	0.37
KRAS	-0.12	None	-0.40
NRAS	0.25	None	-0.45
RRAS	-0.22	None	0.03

Spearman

genes	PRKCA	PRKCB	PRKCG
RRAS2	-0.08	None	0.10
MRAS	0.33	None	0.04
HRAS	0.32	None	0.21
KRAS	0.10	None	-0.01
NRAS	0.13	None	0.08
RRAS	-0.19	None	-0.25

PTPN7, PTPRR, PTPN5 * MAPK14, MAPK11, MAPK13, MAPK12

Pearson

genes	PTPN7	PTPRR	PTPN5
MAPK14	0.19	-0.26	None
MAPK11	-0.27	0.17	None
MAPK13	0.10	0.09	None
MAPK12	-0.10	-0.01	None

Spearman

genes	PTPN7	PTPRR	PTPN5
MAPK14	0.16	-0.14	None
MAPK11	0.13	0.60	None
MAPK13	0.10	0.07	None
MAPK12	0.12	0.06	None

PTPN7, PTPRR, PTPN5 * MAPK1, MAPK3

Pearson

genes	MAPK1	MAPK3
PTPN7	0.01	-0.05
PTPRR	-0.06	-0.29
PTPN5	None	None

Spearman

genes	MAPK1	MAPK3
PTPN7	-0.03	-0.34
PTPRR	0.09	-0.06
PTPN5	None	None

PTPN7, PTPRR, PTPN5 * MAPK8, MAPK9, MAPK10

Pearson

genes	MAPK8	MAPK9	MAPK10
PTPN7	0.17	-0.18	-0.26
PTPRR	0.28	0.37	0.38
PTPN5	None	None	None

Spearman

genes	MAPK8	MAPK9	MAPK10
PTPN7	0.18	-0.19	-0.06
PTPRR	0.11	0.21	0.36
PTPN5	None	None	None

RAC1, RAC2, RAC3, CDC42 * PAK1, PAK2

Pearson

genes	PAK1	PAK2
RAC1	0.09	0.14
RAC2	0.40	0.72
RAC3	0.07	-0.32
CDC42	0.67	0.68

Spearman

genes	PAK1	PAK2
RAC1	0.15	-0.03
RAC2	0.43	0.82
RAC3	-0.02	-0.33
CDC42	0.59	0.77

RASGRF1, RASGRF2 * RRAS2, MRAS, HRAS, KRAS, NRAS, RRAS

Pearson

genes	RASGRF1	RASGRF2
RRAS2	0.01	None
MRAS	0.18	None
HRAS	-0.20	None
KRAS	-0.32	None
NRAS	0.42	None
RRAS	-0.17	None

Spearman

genes	RASGRF1	RASGRF2
RRAS2	0.39	None
MRAS	0.05	None
HRAS	0.07	None
KRAS	0.05	None
NRAS	0.17	None
RRAS	-0.08	None

RASGRP1, RASGRP2, RASGRP4, RASGRP3 * RRAS2, MRAS, HRAS, KRAS, NRAS, RRAS

Pearson

genes	RASGRP1	RASGRP2	RASGRP4	RASGRP3
RRAS2	-0.21	-0.45	None	-0.23
MRAS	-0.19	0.08	None	0.06
HRAS	-0.02	-0.18	None	-0.11
KRAS	0.51	0.01	None	-0.21
NRAS	0.05	0.12	None	0.33
RRAS	-0.18	0.04	None	-0.28

Spearman

genes	RASGRP1	RASGRP2	RASGRP4	RASGRP3
RRAS2	-0.11	-0.15	None	-0.06
MRAS	-0.15	0.12	None	0.06
HRAS	-0.10	-0.16	None	0.10
KRAS	0.34	0.12	None	-0.10
NRAS	0.04	-0.04	None	-0.04
RRAS	-0.14	0.12	None	-0.15

SOS1, SOS2 * RRAS2, MRAS, HRAS, KRAS, NRAS, RRAS

Pearson

genes	SOS1	SOS2
RRAS2	0.36	0.13
MRAS	-0.27	-0.23
HRAS	-0.23	-0.51
KRAS	0.22	0.30
NRAS	0.20	0.50
RRAS	-0.12	-0.08

Spearman

genes	SOS1	SOS2
RRAS2	0.61	0.06
MRAS	-0.34	-0.12
HRAS	-0.15	-0.15
KRAS	0.15	0.28
NRAS	0.47	0.03
RRAS	-0.30	-0.49

TGFB1, TGFB2, TGFB3 * TGFBR1, TGFBR2

Pearson

genes	TGFBR1	TGFBR2
TGFB1	-0.12	0.49
TGFB2	0.26	-0.11
TGFB3	0.09	-0.08

Spearman

genes	TGFBR1	TGFBR2
TGFB1	0.01	0.48
TGFB2	0.36	-0.18
TGFB3	0.13	0.08

b)Resultados de coevolución

Las tablas que siguen representan los datos calculados de coevolución. Se representa la distancia entre los árboles calculados para cada gen mediante el programa treeKO. La mayor semejanza se corresponde con una distancia de 0 (árboles idénticos) y la menor

semejanza con 1. Se representan mediante verde y negro respectivamente como se indica en la escala facilitada.

Los datos ausentes se han representado mediante “None” si no se ha encontrado un árbol adecuado para dicho gen y mediante “treeKO” si el programa de cálculo de distancias ha dado un error en vez de un resultado.

CACNG3, CACNG2, CACNG5, CACNG4, CACNA2D3, CACNG8, CACNG7, CACNG6, CACNA1A, CACNA1B, CACNA1C, CACNA1D, CACNA1E, CACNA1F, CACNA1S, CACNA2D1, CACNB1, CACNB2, CACNB3, CACNB4, CACNG1, CACNA1I, CACNA1H, CACNA1G, CACNA2D2, CACNA2D4 * RASGRF1, RASGRF2

No se han encontrado datos válidos.

CACNG3, CACNG2, CACNG5, CACNG4, CACNA2D3, CACNG8, CACNG7, CACNG6, CACNA1A, CACNA1B, CACNA1C, CACNA1D, CACNA1E, CACNA1F, CACNA1S, CACNA2D1, CACNB1, CACNB2, CACNB3, CACNB4, CACNG1, CACNA1I, CACNA1H, CACNA1G, CACNA2D2, CACNA2D4 * RASGRP1, RASGRP2, RASGRP4, RASGRP3

No se han encontrado datos válidos.

CHUK, IKBKB, IKBKG * NFKB1, NFKB2, RELA, RELB

No se han encontrado datos válidos.

DUSP14, DUSP10, DUSP1, DUSP2, DUSP3, DUSP4, DUSP5, DUSP6, DUSP7, DUSP8, DUSP9, DUSP22, DUSP16 * MAPK14, MAPK11, MAPK13, MAPK12

genes	MAPK14	MAPK11	MAPK13	MAPK12
DUSP14	None	None	None	None
DUSP10	None	None	None	None
DUSP1	None	None	None	None
DUSP2	None	None	None	None
DUSP3	None	None	None	None
DUSP4	treeKO	treeKO	treeKO	treeKO
DUSP5	None	None	None	None
DUSP6	0.71	0.80	0.78	0.80
DUSP7	None	None	None	None
DUSP8	0.54	0.65	0.65	0.61
DUSP9	None	None	None	None
DUSP22	None	None	None	None
DUSP16	0.68	0.60	0.72	0.60

DUSP14, DUSP10, DUSP1, DUSP2, DUSP3, DUSP4, DUSP5, DUSP6, DUSP7, DUSP8, DUSP9, DUSP22, DUSP16 * MAPK1, MAPK3

genes	MAPK1	MAPK3
DUSP14	None	None
DUSP10	None	None
DUSP1	None	None
DUSP2	None	None
DUSP3	None	None
DUSP4	treeKO	treeKO
DUSP5	None	None
DUSP6	0.87	0.68
DUSP7	None	None
DUSP8	0.75	0.61
DUSP9	None	None
DUSP22	None	None
DUSP16	0.82	0.82

DUSP14, DUSP10, DUSP1, DUSP2, DUSP3, DUSP4, DUSP5, DUSP6, DUSP7, DUSP8, DUSP9, DUSP22, DUSP16 * MAPK8, MAPK9, MAPK10

No se han encontrado datos válidos.

FGF1, FGF2, FGF3, FGF4, FGF5, FGF6, FGF7, FGF8, FGF9, FGF10, FGF11, FGF12, FGF13, FGF14, FGF20, FGF21, FGF22, FGF23, FGF18, FGF17, FGF16, FGF19 * FGFR1, FGFR3, FGFR2, FGFR4

genes	FGFR1	FGFR3	FGFR2	FGFR4
FGF1	None	None	None	None
FGF2	None	None	None	0.80
FGF3	None	None	None	0.83
FGF4	None	None	None	0.80
FGF5	None	None	None	None
FGF6	None	None	None	None
FGF7	None	None	None	0.82
FGF8	None	None	None	None
FGF9	None	None	None	None
FGF10	None	None	None	0.84
FGF11	None	None	None	None
FGF12	None	None	None	0.84
FGF13	None	None	None	None
FGF14	None	None	None	None
FGF20	None	None	None	None
FGF21	None	None	None	0.76
FGF22	None	None	None	0.72
FGF23	None	None	None	0.77
FGF18	None	None	None	None
FGF17	None	None	None	None
FGF16	None	None	None	None
FGF19	None	None	None	0.73

HSPA1A, HSPA1B, HSPA1L, HSPA2, HSPA6, HSPA8 * MAPK8, MAPK9, MAPK10

No se han encontrado datos válidos.

IL1A, IL1B * IL1R1, IL1R2

No se han encontrado datos válidos.

MAPK14, MAPK11, MAPK13, MAPK12 * MAPKAPK3, MAPKAPK2

No se han encontrado datos válidos.

MAPK14, MAPK11, MAPK13, MAPK12 * RPS6KA4, RPS6KA5

genes	RPS6KA4	RPS6KA5
MAPK14	0.92	0.80
MAPK11	0.73	0.65
MAPK13	0.63	0.77
MAPK12	0.86	0.77

MAPK1, MAPK3 * MKNK2, MKNK1

genes	MKNK2	MKNK1
MAPK1	0.67	0.84
MAPK3	0.57	0.71

MAPK1, MAPK3 * PLA2G4B, PLA2G4E, PLA2G2D, PLA2G2E, PLA2G2C, PLA2G3, PLA2G1B, PLA2G2A, PLA2G4A, PLA2G5, PLA2G2F, PLA2G12A, PLA2G6, PLA2G10, PLA2G12B, JMJD7-PLA2G4B

No se han encontrado datos válidos.

MAPK1, MAPK3 * RPS6KA6, RPS6KA1, RPS6KA2, RPS6KA3

genes	MAPK1	MAPK3
RPS6KA6	0.75	0.59
RPS6KA1	0.77	0.71
RPS6KA2	0.82	0.67
RPS6KA3	0.94	0.68

MAPK8IP2, MAPK8IP1 * MAPK8, MAPK9, MAPK10

No se han encontrado datos válidos.

MAPK8, MAPK9, MAPK10 * CRK, CRKL

No se han encontrado datos válidos.

NTF3, NTF4 * NTRK1, NTRK2

No se han encontrado datos válidos.

PAK1, PAK2 * MAP3K2, MAP3K3

No se han encontrado datos válidos.

PDGFA, PDGFB * PDGFRA, PDGFRB

genes	PDGFRA	PDGFRB
PDGFA	None	None
PDGFB	0.79	0.33

PRKACA, PRKACB, PRKACG, PRKX * RAP1A, RAP1B

genes	RAP1A	RAP1B
PRKACA	treeKO	0.78
PRKACB	None	None
PRKACG	treeKO	0.81
PRKX	treeKO	0.77

PRKCA, PRKCB, PRKCG * RRAS2, MRAS, HRAS, KRAS, NRAS, RRAS

genes	PRKCA	PRKCB	PRKCG
RRAS2	0.68	0.79	0.57
MRAS	0.70	0.73	0.68
HRAS	0.89	0.71	0.65
KRAS	1.00	1.00	1.00
NRAS	0.92	0.84	0.88
RRAS	0.88	0.77	0.83

PTPN7, PTPRR, PTPN5 * MAPK14, MAPK11, MAPK13, MAPK12

No se han encontrado datos válidos.

PTPN7, PTPRR, PTPN5 * MAPK1, MAPK3

No se han encontrado datos válidos.

PTPN7, PTPRR, PTPN5 * MAPK8, MAPK9, MAPK10

No se han encontrado datos válidos.

RAC1, RAC2, RAC3, CDC42 * PAK1, PAK2

No se han encontrado datos válidos.

RASGRF1, RASGRF2 * RRAS2, MRAS, HRAS, KRAS, NRAS, RRAS

genes	RASGRF1	RASGRF2
RRAS2	1.00	0.77
MRAS	1.00	0.76
HRAS	0.79	0.66
KRAS	1.00	1.00
NRAS	1.00	0.86
RRAS	0.62	1.00

RASGRP1, RASGRP2, RASGRP4, RASGRP3 * RRAS2, MRAS, HRAS, KRAS, NRAS, RRAS

No se han encontrado datos válidos.

SOS1, SOS2 * RRAS2, MRAS, HRAS, KRAS, NRAS, RRAS

genes	SOS1	SOS2
RRAS2	0.51	0.54
MRAS	0.61	0.62
HRAS	0.67	0.61
KRAS	0.88	1.00
NRAS	0.67	0.80
RRAS	1.00	1.00

TGFB1, TGFB2, TGFB3 * TGFBR1, TGFBR2

genes	TGFBR1	TGFBR2
TGFB1	0.59	treeKO
TGFB2	0.81	treeKO
TGFB3	0.79	treeKO

10.3. Apéndice III. CÓDIGO FUENTE DE PROGRAMAS

El código fuente de los programas se facilita en ficheros electrónicos.

a)Programa principal para el cálculo de la coexpresión

coexpression.py

```
#!/usr/bin/env python

# IMPORTS:
import sys
import pickle
#from datetime import datetime
#import subprocess
#import collections
# 3rd party:
from rpy2 import robjects as R
# Own modules:
import utilities as utils
from bidimensional_array import BidimensionalArray as array

# HARDCODES FOR PATHS AND FILENAMES:
work_dir = "/home/viridis/Documentos/POP-fin/data/Trab-DANIEL/Trab-
DANIEL/"
utils.ensure_dir(work_dir+'newresults/')
#gene_names = open(work_dir+"mxNAMES-hgu133a-22283gp.txt")

# Arguments preparation:
# Correlation methods to use:
correlations = ("pearson","spearman")

## Class definitions:
class Gene(object):
    """Class to store data of gene expression."""
    def __init__(self,name,affy_ids,altnames,expression):
        """
        Init method
        name: name of the gene
        affy_id: ids of Affymetrix for this gene. Might be more than
one
        altnames: list of alternative names
        expression: list of values found in the data file
        """
        self.name = name
        self.affy_ids = affy_ids
        self.altnames = altnames
        self.expression = expression

    def is_same_gene(self,query):
        """Method to look if a gene name is this (considering
alternative names)"""
        if query == self.name or query in altnames:
            return True
        # Maybe a wraping character exists, such as quotes, etc:
        myquery = query[1:-1]
        if myquery == self.name or myquery in altnames:
            return True
```

```
    def is_in_line(self,line):
        """Checks if any of the stored gene names is in line"""
        if '"'+self.name+'"' in line: return True
        for synonym in self.altnames:
            if '"'+synonym+'"' in line: return True
        return False

    def has_affy_id(self,query):
        """Method to look if an ID of Affymetrix points to this gene
or not"""
        return query in self.affy_ids

    def add_expression_values(self,key,expression_list):
        """Updates the dictionary of expression for this gene"""
        # Might be text that needs to be converted:
        float_expression = [float(datum) for datum in
expression_list]
        try:    # The dictionary entry might be already in use:
            existing_expressions = self.expression[key]
        except KeyError:    #... or maybe is not:
            self.expression.update({key:[float_expression]})
            # So an append is needed (to average at last)
        else: existing_expressions.append(float_expression)

    def average_out_expression_data(self):
        """Method to calculate the mean of each self.expression
entry. Note that used to be a list of lists."""
        # Small function to average out items:
        average = lambda *items: sum(items)/len(items)
        for key,value in self.expression.iteritems():
            if not isinstance(value[0],list): continue
            self.expression.update({key:map(average,*value)})
            print key, value

    def robject_expression_floatvector(self,key):
        """Method to build a robject vector (via rpy2) with the data
stored in expression"""
        try:
            vector = R.FloatVector(self.expression[key])
        except KeyError:
            return None
        else:
            return vector

    def calculate_correlation(self,other,method):
        """Calculates the correlation with rpy2 given an other gene
object and the desired method"""
        v1 = self.robject_expression_floatvector(method)
        v2 = other.robject_expression_floatvector(method)
        if v1 is None or v2 is None: return None
        # The correlation function cor returns a vector of 1 result:
        else: return R.r.cor(v1,v2)[0]

    def iter_all_names(self):
        """Iterator of the names (including known synonims)"""
        for name in self.altnames + [self.name]:
            yield name

################ Start of main script:
#The names of the genes to be processed:
try:
```

```
    genes1 = [Gene(name,[],[],{}) for name in
sys.argv[1].split(",")]
    genes2 = [Gene(name,[],[],{}) for name in
sys.argv[2].split(",")]
    allgenes = sys.argv[1].split(",") + sys.argv[2].split(",")
except IndexError:
    print 'Missing commandline options'
    print sys.argv[0], ' geneslist1 genelist2 [genesaltnames]'
    print 'geneslist1: list of genes separated with commas (no
spaces)'
    print 'geneslist2: list of genes separated with commas (no
spaces)'
    exit()

# Structure with expression data retrieval. Is a named tuple with
dictionaries:
# affy_to_genename
# genename_to_affy
# affy_to_data_rma
# affy_to_data_mas5
affy_to_genename = pickle.load(open('affy_to_genename','rb'))
genename_to_affy = pickle.load(open('genename_to_affy','rb'))
affy_to_data_rma = pickle.load(open('affy_to_data_rma','rb'))
affy_to_data_mas5 = pickle.load(open('affy_to_data_mas5','rb'))
#expression = pickle.load(open('expression_data','rb'))

# Trying to put the data in place:
# Pearson
# genes1
for gene in genes1:
    for name in gene.iter_all_names():
        try:
            ids = genename_to_affy[name]
        except KeyError:
            continue
        else:
            gene.affy_ids = ids
            for affy_id in ids:
                gene.add_expression_values('pearson',affy_to_data_rm
a[affy_id])
# genes2
for gene in genes2:
    for name in gene.iter_all_names():
        try:
            ids = genename_to_affy[name]
        except KeyError:
            continue
        else:
            gene.affy_ids = ids
            for affy_id in ids:
                gene.add_expression_values('pearson',affy_to_data_rm
a[affy_id])
# Spearman
# genes1
for gene in genes1:
    for name in gene.iter_all_names():
        try:
            ids = genename_to_affy[name]
        except KeyError:
            continue
        else:
            gene.affy_ids = ids
            for affy_id in ids:
```

```
                gene.add_expression_values('spearman',affy_to_data_m
as5[affy_id])
# genes2
for gene in genes2:
    for name in gene.iter_all_names():
        try:
            ids = genename_to_affy[name]
        except KeyError:
            continue
        else:
            gene.affy_ids = ids
            for affy_id in ids:
                gene.add_expression_values('spearman',affy_to_data_m
as5[affy_id])

# Average out method must be invoked to have one single list per
gene and method:
for gene in genes1:
    gene.average_out_expression_data()
for gene in genes2:
    gene.average_out_expression_data()

#-----------------------------------------------------------------
-----------#
## All data needed has been already retrieven:
# Both correlations:
for method in correlations:
    # The array of data is prepared with column names:
    results = [['genes']+[gene.name for gene in genes2]]
    # Nested loop over the genes:
    for g1 in genes1:
        # Each new row will start with the gene name
        results.append([g1.name])
        for g2 in genes2:
            correlation = g1.calculate_correlation(g2,method)
            # Calculated correlation corresponds to the last
(current) gene:
            results[-1].append(correlation)
    # A proper array is created:
    print results
    results = array(results)
    results.sort_rows_and_columns()
    results.make_vertical()
    results.export_to_text_tab(work_dir+'newresults/'+
        '_'.join(sys.argv[1].split(","))+'+'+
        '_'.join(sys.argv[2].split(","))+'-'+method)
```

b)Programa para la descarga de datos de KEGG y serialización en local para los cálculos de coexpresión.

coexpression_download_kegg_data.py

```
#!/usr/bin/env python

# For MAPK use map_id hsa04010

# IMPORTS
# Standard modules:
import os
import sys
```

```
# Own modules:
import kegg_classes as kegg
import utilities as utils

# Captures the arguments from commandline:
try:
    map_id = sys.argv[1]
    local_dir = sys.argv[2] if sys.argv[2].endswith('/') else
sys.argv[2]+'/'
except IndexError:
    print 'Commandline arguments missing.\nUsage:'
    print sys.argv[0] + ' <KEGG_map_id>
<local_path_to_retrieve_or_save_data>'
    print '\nPlease use quotation marks if path contains spaces.'
    exit()

# Ensures the existance of (or creates) the directory
utils.ensure_dir(local_dir)

# Mymap object for map of interest
mymap = kegg.KeggMap(map_id, filter_element_type='gene',
filter_min_subelements = 2, local_dir = local_dir, local_save =
True)

# Dictionary for alternative gene names:
altgenes = {}

# Saves the data to a file with the structure:
# geneA1,geneA2,...,geneAn geneB1,geneB2,...,geneBn
with open(local_dir + 'genes_for_coexpression','w') as genes:
    for knode1, knode2, krelation in
mymap.iter_elements_relations():
        # Lists of names of genes. First option is always used:
        print knode1
        print knode2
        if knode1 is None or knode2 is None: continue

        knames1 = [ksubnode.names[0] for ksubnode in
knode1.subnodes]
        knames2 = [ksubnode.names[0] for ksubnode in
knode2.subnodes]
        genes.write(','.join(knames1) + ' ' +
','.join(knames2)+'\n')

# Alternative names stored:
with open(local_dir + 'genes_for_coexpression_altnames','w') as
genes:
    for knode in mymap.elements:
        # Lists of names of genes. First option is always used:
        if knode is None: continue
        for ksubnode in knode.subnodes:
            genes.write('\t'.join(ksubnode.names)+'\n')
```

c) Programa para la recuperación de datos de expresión.

retieve_expression_data.py

```
#!/usr/bin/env python

# Imports
#from collections import namedtuple
```

```
import pickle

# HARDCODES FOR PATHS AND FILENAMES:
work_dir = "/home/viridis/Documentos/POP-fin/data/Trab-DANIEL/Trab-
DANIEL/"

names = []

with open('genes_for_coexpression_altnames','r') as genenames:
    for line in genenames:
        for item in line[:-1].split('\t'):
            names.append(item)
#    names = [line[:-1].split('\t') for line in genenames]

affy_to_genename = {}

with open(work_dir+"mxNAMES-hgu133a-22283gp.txt") as genenames:
    for line in genenames:
        for gene in names:
            if '"'+gene+'"' in line:
                affy_to_genename.update({line.split('\t')[0]:gene})
#                genename_to_affy.update({gene:line.split('\t')[0]})

genename_to_affy = {}
for k,v in affy_to_genename.iteritems():
    try:
        genename_to_affy[v].append(k)
    except KeyError:
        genename_to_affy.update({v:[k]})

affy_to_data_mas5 = {}  # For spearman correlation
affy_to_data_rma = {}   # For pearson correlation

with open(work_dir+"mx48tissue-rma-Allgp.txt") as datafile:
    for line in datafile:
        dataline = line[:-1].split('\t')
        try:
            affy_to_genename[dataline[0]]
        except KeyError:
            continue
        else:
            affy_to_data_rma.update({dataline[0]:dataline[1:]})

with open(work_dir+"mx48tissue-mas5-Allgp.txt") as datafile:
    for line in datafile:
        dataline = line[:-1].split('\t')
        try:
            affy_to_genename[dataline[0]]
        except KeyError:
            continue
        else:
            affy_to_data_mas5.update({dataline[0]:dataline[1:]})

# Data structure to save it to file:
#expression_struct = namedtuple('expression','affy_to_genename
genename_to_affy affy_to_data_rma affy_to_data_mas5')
#struct =
expression_struct(affy_to_genename,genename_to_affy,affy_to_data_rma
,affy_to_data_mas5)
pickle.dump(affy_to_genename, open('affy_to_genename','wb'))
pickle.dump(genename_to_affy, open('genename_to_affy','wb'))
pickle.dump(affy_to_data_rma, open('affy_to_data_rma','wb'))
pickle.dump(affy_to_data_mas5, open('affy_to_data_mas5','wb'))
```

```
#pickle.dump(struct,open('expression_data','wb'))
```

d)Programa para la ejecución del programa de coexpresión para una lista de genes generada previamente.

execute_coexpression_genes_list.py

```
#!/usr/bin/env python

# Might use:
# python execute_coexpression_genes_list.py coexpression.py
genes_for_coexpression

# Be careful, if any of the executions of coexpression.py raises an
exception, it will not stop this program and might go unnoticed

# Imports:
import sys
import subprocess

# Commandline arguments processing:
try:
    coexpression_program = sys.argv[1]
    geneslist = sys.argv[2]
except IndexError:
    print 'Commandline options missing.'
    print 'Usage: ', sys.argv[0], ' coexpression_program
geneslist_path [altnames_genes]'
    print 'coexpression_program: path to the coexpression python
program'
    print 'geneslist_path: path to a file with the geneslist
generated by the program "coexpression_download_kegg_data.py"'
    print 'altnames_genes (optional): file with alternative names
for genes'
    exit()

try:
    altnames = sys.argv[3]
except IndexError: altnames = ''

# Opens the file with the gene families prepared and iterates over
the data
with open(geneslist,'r') as genes:
    #each line is a pair of gene families represented by their names
in KEGG
    for gene_group in genes:
        gene_fams = gene_group.split()
        print 'PROCESSING GENES... ', gene_fams[0], gene_fams[1]
        # System call
        out = subprocess.Popen([
            "python",
            coexpression_program,
            gene_fams[0],
            gene_fams[1],
            altnames
            ], stdout=subprocess.PIPE).communicate()[0]
        print 'OUTPUT FROM COEXPRESSION PROGRAM: ',out
```

e)Programa para el cálculo de la coevolución.

coevolution.py

```
#!/usr/bin/env python


# Standard modules:
import os
import sys
# 3rd party modules:
import ete2
# Own modules:
import kegg_classes2 as kegg
import utilities as utils
from bidimensional_array import BidimensionalArray as array

# Recovers the kegg map id from the commandline
try:
    kegg_map = sys.argv[1]
except IndexError:
    print 'Missing commandline options'
    print 'Usage:'
    print sys.argv[0], '<kegg map id>'

# Results directory:
work_dir = 'working_directory_coevolution/'

# Serialization directory:
mylocal_dir = 'serialized_coevo/'
utils.ensure_dir(mylocal_dir)

# Mymap object for map of interest
mymap =
kegg.KeggMap(kegg_map,filter_element_type='gene',filter_min_subeleme
nts=2,local_dir = mylocal_dir,local_save = True)

# Abbreviation
m = mymap

# creates the dictionary crossreferences and populates it with the
keys to look for
crossrefs = {}
for n in m.elements:
    if n.type != 'gene': continue
    for sn in n.subnodes:
        for db,crossref_list in sn.dblinks.items():
            crossrefs.update([[crossref,None] for crossref in
crossref_list])

# Look for PhylomeDB IDs on file
with open('phylomedb/all_id_conversions') as ids:
    for line in ids:
        line_data = line.split('\t')
        crossref = line_data[-1][:-1]
        try: crossrefs[crossref]
        except KeyError: pass
        else: crossrefs.update({crossref:line_data[0]})

# Update the subnodes phyprots set attribute with the actual
PhylomeDB ids
for n in m.elements:
    if n.type != 'gene': continue
    for sn in n.subnodes:
```

```
        sn.phyprots = []
        for db,crossref_list in sn.dblinks.items():
            for crossreference in crossref_list:
                sn.phyprots.append(crossrefs[crossreference])    # 4
version changed from set to list

# Dictionary to store all the best trees (PhylomeDB) for each gene
entry in the map.
phydb_dict = {}
phydb_dict.update([[phyprot,None] for phyprot in crossrefs.values()
if phyprot])

with open('phylomedb/best_trees.txt') as bt:
    for line in bt:
        line_data = line.split('\t')
        try: phydb_dict[line_data[0]]
        except KeyError: pass
        else: phydb_dict.update({line_data[0]:line_data[3][:-1]})
# PhylomeDB trees update in the KeggMap structure (all possible
trees for node without filtering).
for n in m.elements:
    for sn in n.subnodes:
        n.phytrees = []
        for phyprot in sn.phyprots:
            try: tree_str = phydb_dict[phyprot]
            except KeyError: pass
            else: n.phytrees.append(ete2.Tree(tree_str))

######################################################################
##################
# Prepares the result directory:
utils.ensure_dir(work_dir+'coevo_newresults/')

# General loop to check and compare data
for n1,n2,r in m.iter_elements_relations():
    # Checks if there is data:
    if n1 and n1.phytrees and n2 and n2.phytrees:
        subtrees_phyprots1 = n1.subtrees_single_subnode()
        subtrees_phyprots2 = n2.subtrees_single_subnode()
        if not subtrees_phyprots1 or not subtrees_phyprots2:
continue
        row_header = ['genes']+[subnode.names[0] for subnode in
[phyprot.values()[0] for subtree,phyprot in subtrees_phyprots2 ]]
        col_header = ['genes']+[subnode.names[0] for subnode in
[phyprot.values()[0] for subtree,phyprot in subtrees_phyprots1 ]]
        results = [row_header]
        for subtree1,phyprot1 in subtrees_phyprots1:
            results.append([phyprot.values()[0]])
            for subtree2,phyprot2 in subtrees_phyprots2:
                distance =
utils.calculate_distance_treeko(subtree1,subtree2)
                results[-1].append(float(distance))
        # Now the result array is built:
        results = array(results)
        results.make_vertical()
        results.export_to_text_tab(work_dir+'coevo_newresults/'+
            '_'.join(row_header) + '+' +
            '_'.join(col_header))
```

f) Programa para la gestión de matrices bidimensionales para el formateo de resultados.

bidimensional_array.py

```
#!/usr/bin/env python

# IMPORTS:
from itertools import count

"""
# Test with this list:
l=[]
l.append(['rows\/cols','c1','c2','c3','c4'])
l.append(['r1',0,1,2,3])
l.append(['r2',0,1,2,3])
l.append(['r3',0,1,2,3])
"""

class BidimensionalArray(object):
    def __init__(self,list_of_lists):
        """
        Init method. Takes a list_of_lists argument given the first
list in
        the list_of_lists is the column names and the first item in
each list is
        the row name.
        """
        assert list_of_lists, 'Given array is empty'
        assert all(list_of_lists), 'Given array is empty'
        item_count = len(list_of_lists[0])
        for l in list_of_lists:
            assert len(l) == item_count, 'Array is inconsistent, all
lists must be the same lenght' + str(self)
        self.list_of_lists = list_of_lists

    def transpose(self):
        """
        Transpose the array, that is, inverts the colums with the
rows and
        viceversa.
        """
        new_array = [list(atuple) for atuple in
zip(*self.list_of_lists)]
        self.list_of_lists = new_array

    def is_square(self):
        """Returns boolean value if colums == rows"""
        return len(list_of_lists) == len(list_of_lists[0])

    def sort_rows_and_columns(self):
        """Sort both dimensions"""
        self.sort_rows_by_column
        self.sort_columns_by_row

    def sort_rows_by_column(self,column=0,first_is_header=True):
        """Sorts the array by the given column"""
        new_array = []
        unsorted = [row[column] for row in self.list_of_lists]
```

```
            if first_is_header: unsorted.pop(0)
            mysorted = sorted(unsorted)

            indices = [mysorted.index(item) for item in unsorted]

            for number in count(1 if first_is_header else 0):
                try:
                    new_array.append(unsorted[indices.index(number)])
                except IndexError:
                    break

            if first_is_header:
                new_array.insert(0,list_of_lists[0])

            self.list_of_lists = new_array

        def sort_columns_by_row(self,row=0,first_is_header=True):
            """Sorts the array by the given row"""
            new_array = [[]] * len(list_of_lists)
            unsorted = self.list_of_lists[0]
            if first_is_header: unsorted.pop(0)
            mysorted = sorted(unsorted)

            for i,row in enumerate(list_of_lists):
                if first_is_header: new_array[i].append(row[0])
                for item in mysorted:
                    if first_is_header:
                        new_array[i].append(row[unsorted.index(item)+1])
                    else:
                        new_array[i].append(row[unsorted.index(item)])
            self.list_of_lists = new_array

        @property
        def columns(self):
            """The number of rows"""
            return len(self.list_of_lists[0])
        @property
        def rows(self):
            """The number of rows"""
            return len(self.list_of_lists)

        def make_vertical(self):
            """Makes the array vertical (columns more than rows)"""
            if self.columns > self.rows:
                self.transpose()

        def make_horizontal(self):
            """Makes the array horizontal (rows more than columns)"""
            if self.columns < self.rows:
                self.transpose()

        def
    lines_iterator(self,separator='\t',decimal_places_for_numbers=2):
            """Iterator for the rows. Yields formatted lines"""
            for row in self.list_of_lists:
                line = []
                for item in row:
                    try:
                        fitem = float(item)
                    except ValueError:
                        line.append(item)
                    except TypeError:
                        line.append(item)
```

```
                else:
                    line.append(eval('"%.'+str(decimal_places_for_nu
mbers)+'f" % fitem'))
            yield separator.join(str(item) for item in line)

    def
export_to_text_tab(self,path='table',decimal_places_for_numbers=2):
        """Exports the array to a file in tab-delimited format"""
        with open(path,'w') as f:
            f.write('\n'.join(list(self.lines_iterator('\t',decimal_
places_for_numbers))))

    def __str__(self):
        """String conversion method"""
        return '\n'.join(list(self.lines_iterator()))
```

g)Programa de generación de matrices de falso color rojo-verde (macro de ImageJ)

Este programa está escrito en el lenguaje de macros de ImageJ. Los detalles pueden consultarse en la siguiente dirección web: http://rsbweb.nih.gov/ij/developer/index.html.

dir_matrix2color.ijm

```
// With directory support
dir_in = getDirectory("Select input text files directory...");
dir_out = getDirectory("Select output matrices directory...");
file_lst = getFileList(dir_in);
setBatchMode(true);

for (i=0;i<lengthOf(file_lst);i++) {
       showProgress(i/lengthOf(file_lst));
       print("processing",file_lst[i]);
       color_matrix(dir_in,dir_out,file_lst[i]);
}
waitForUser("Finished!");

function color_matrix(dir_in,dir_out,filename) {
       run("Text Image... ", "open="+dir_in+filename);
       // For an unknown reason, the first row disappears, so canvas
adjust is necessary:
       newHeight = getHeight()+1;
       run("Canvas Size...", "width="+getWidth()+"
height="+newHeight+" position=Bottom-Center zero");
       data = getImageID();
// Sets the maximum zoom:
       run("Set... ", "zoom=3200");
// Again the zoom is set to maximum:
       run("Set... ", "zoom=3200");
// The image is 32-bit values (grey) so we need an image the same
size to fill with colors:
       run("Duplicate...", "title="+filename);
       run("Set... ", "zoom=3200");
       run("RGB Color");
       color = getImageID();
////////////// Now 2 images are opened --------------------

       for (x=0; x<getWidth(); x++) {
              for (y=0; y<getHeight(); y++) {
```

```
                        selectImage(data);
                        px_value = getPixel(x,y);
                        if (px_value > 0) { // GREEN
                                RGB_value = "00"+toHex(255*px_value)+"00";
        //green if positive
                        }
                        else {              // RED or BLACK
                                RGB_value = toHex(abs(255*px_value))
+"00"+"00";   //red if negative
                        }
//                      print(RGB_value);
//                      print(parseInt(RGB_value,16));
                        selectImage(color);
                        setPixel(x,y,parseInt(RGB_value,16));
        }
}

if (getWidth() != 0 && getHeight != 0) {
        saveAs("PNG", dir_out+filename+".png");
        run("Scale...", "x=32 y=32 interpolation=None create
title=zoom");
        saveAs("PNG", dir_out+"zoom/"+filename+".png");
        close();
}
        else print("Width or Height = 0 found");
close();
close();
} //end of function color_matrix
```

h)Utilities in various functions

utilities.py

```
#!/usr/bin/env python

# Standard modules:
import os
import pickle

def ensure_dir(mypath):
        """Creates directories if the path does not exist"""
        d = os.path.dirname(mypath)
        if not os.path.exists(d):
            os.makedirs(d)

def pickle_anything(object2pickle, relative_path,local_dir = None):
    """ This method is an aid to pickle anything easily within this
object. Caution, it will overwrite any existing file. Checks if
local save is true"""
    if local_dir: path = local_dir + relative_path
    else: path = relative_path
    ensure_dir(path)
    pickling_file = open(path,  'w')
    pickle.dump(object2pickle,  pickling_file)
    pickling_file.close()

def unpickle_anything(relative_path, local_dir = None):
    """Method to unpickle objects within this instance."""
#        unpickled = None
    if local_dir: path = local_dir + relative_path
    else: path = relative_path
```

```
    try:
        pickle_file = open(path, 'r')
        unpickled = pickle.load(pickle_file)
    except IOError:
         unpickled = None
#         if relative_path.endswith('2.pck'): raise IOError
    return unpickled

def phylomedb_filter_human_prot(*protids):
    """Takes one or more PhylomeDB IDs and filters only human
(specified by 'Hsa' prefix)"""
    filtered_ids = []
    for id in protids:
        if id.startswith('Hsa'):    # Does not work with DB3
            filtered_ids.append(id)
    return filtered_ids

def calculate_distance_treeko(tree1, tree2,
treeKO_path='treeKO/treeKO.py', treeKO_prog='tc', config_file=None,
distance='strict', temporary_path='tmp/'):
    """This function simplifies the call to treeKO tree comparison
program. It returns the specified distance. Trees or files may be
given as input."""
    import subprocess

    def get_tree_filepath(tree, filename = 'tree.tmp',
temporary_path='tmp/'):
        """This function gets a tree object or a file_path and
returns the filepath for this tree."""
        from os import path
        from ete2 import  Tree

        if isinstance(tree, Tree):
            tree.write(outfile=temporary_path+filename)
            return temporary_path+filename
        elif isinstance(tree, str):
            if path.isfile(tree):
                return tree
        else: raise Error

    # calculate_distance_treeko
    treeKO_command = ['python',treeKO_path,'-p',treeKO_prog,'-
a',get_tree_filepath(tree1,'tree1.tmp'),'-
b',get_tree_filepath(tree2,'tree2.tmp'),'-c','treeko_config']
    try:
        subprocess.check_call(treeKO_command)
        print 0
    except subprocess.CalledProcessError:
        print 1
        return None
    else: treeko_calc =
subprocess.check_output(treeKO_command).splitlines()
    distance_calc= None
    if distance == 'strict': line2search='The strict distance
between the two trees is:'
    elif distance == 'speciation': line2search='The speciation
distance between the two trees is:'
    else: raise NotImplemented

    print treeko_calc
    print 2
    for line in treeko_calc:
        if line2search in line:
```

```
                print line
                print line[-5:]
                distance_calc = float(line[-5:])
                break
        # Finally the calculated distance is returned:
#        raw_input(distance_calc)
        print 3
        return distance_calc
```

10.4. Apéndice IV. NOTAS PARA PROGRAMADORES

Este apéndice pretende dar una información extendida a las personas que quieran utilizar los programas desarrollados en este proyecto para procesar otros datos o la API diseñada para elaborar sus propios programas.

a)Estilo de programación

- Se ha seguido un estilo mixto entre programación funcional y orientada a objetos.
- Se ha utilizado una indentación de cuatro espacios. Es lo recomendado por la guía oficial de estilo de Python (http://www.python.org/dev/peps/pep-0008/#indentation). Si se mezclan tabulaciones en ficheros con indentaciones mediante espacios, Python no interpretará las líneas nuevas será como si no existieran. Tampoco es recomendable importar módulos indentados de forma distinta. Se recomienda editar el código con un editor que muestre los espacios y tabulaciones como visibles para evitar errores.
- Python se considera un lenguaje de programación que se autodocumenta, es decir, que el propio código debería ser legible como lenguaje humano (en inglés). Para ello conviene que los identificadores (de variables, funciones, clases y métodos) se entiendan por su propio nombre. La sintaxis de Python está diseñada para que sea legible.
- Se han insertado comentarios cuando se consideraban necesarios mediante el comentario de fin de línea, que se especifica mediante el carácter “#”.
- Se ha procurado utilizar cadenas de documentación (*docstrings*), en todas las definiciones de clase, método o funciones. Esto permite la generación de documentación en diversos formatos, como HTML, que se incluye en ficheros adjuntos en este proyecto en el subdirectorio “doc” dentro del directorio de código fuente.

b)Ampliación de los programas

Para la ampliación de los programas de este proyecto se recomienda la subclasificación de las clases diseñadas. Python permite la herencia de clases de un modo muy flexible, de manera que en una clase heredada se pueden eliminar atributos o incluso se pueden añadir o eliminar estos atributos en los objetos instanciados en tiempo de ejecución, de modo dinámico.

No obstante, también se puede modificar el código fuente rediseñando las clases, puesto que la licencia lo permite.

10.5. Apéndice V. LICENCIA DE PROGRAMAS

Todos los programas facilitados se rigen por una licencia BSD simplificada que se reproduce a continuación.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the copyright holder.

10.6. Apéndice VI. ENLACES A PÁGINAS WEB

- **CIC:** http://www.cicancer.org/
- **ETE2:** http://ete.cgenomics.org/
- **ImageJ**: http://rsbweb.nih.gov/ij/
- **KEGG**: http://www.genome.jp/kegg/
- **PhylomeDB**: http://phylomedb.org/
- **R**: http://www.r-project.org/
- **rpy2**: http://rpy.sourceforge.net/rpy2.html
- TreeKO: http://treeko.cgenomics.org/doku.php

10.7. Apéndice VII. ÍNDICE ALFABÉTICO.

APÉNDICES