UNIVERSIDAD NACIONAL DE ROSARIO
FACULTAD DE CIENCIAS AGRARIAS
FACULTAD DE CIENCIAS BIOQUÍMICAS Y FARMACÉUTICAS
INTITUTO DE BIOLOGÍA MOLECULAR Y CELULAR DE ROSARIO

"ANÁLISIS Y MODELADO ESTRUCTURAL DE LOS DOMINIOS CATALÍTICOS DE DCL1 DE *Arabidopsis thaliana*"

Lic. Florencia Carla Mascali

TRABAJO FINAL PARA OPTAR AL TITULO DE ESPECIALISTA EN BIOINFORMÁTICA

DIRECTOR: Dr. Rodolfo M. Rasia

AÑO 2017

ANÁLISIS Y MODELADO ESTRUCTURAL DE LOS DOMINIOS CATALÍTICOS DE DCL1 DE *Arabidopsis thaliana*

Florencia Carla Mascali

Licenciada en Biotecnología – Universidad Nacional de Rosario

Este Trabajo Final es presentado como parte de los requisitos para optar al grado académico de Especialista en Bioinformática, de la Universidad Nacional de Rosario y no ha sido previamente presentada para la obtención de otro título en ésta u otra Universidad. El mismo contiene los resultados obtenidos en investigaciones llevadas a cabo en Instituto de Biología Molecular y Celular de Rosario, durante el período comprendido entre Febrero de 2014 y Junio de 2017, bajo la dirección del Dr. Rodolfo M. Rasia.

Nombre y firma del autor

Nombre y firma del Director

Defendida: .....de 2017.

Agradecimientos

Quiero agradecer a Fito por dirigirme en este trabajo y enseñarme las maravillas de Python, especialmente por permitirme concluir esta formación dentro del espacio de mi Doctorado.

A los chicos de Bioinfo por ayudarme a transitar este camino; por las clases, almuerzos y juntadas llenas de risas: Iva, Mauro y Mauri.

A mi familia por el apoyo constante en favor de mi progreso.

Y dedico este trabajo especialmente a Leo. Gracias por mostrarme el mundo de la informática, sin vos no hubiese querido empezar todo esto. Gracias por ayudarme frente a cada problema que surgía, por bancarme cuando las cosas no iban bien y por siempre estar a mi lado.

Resumen

Los micro-ARNs (miARN) son una clase de ARNs pequeños que participan activamente de procesos biológicos fundamentales tales como el desarrollo de los organismos y las respuestas a cambios ambientales. Su biogénesis es un proceso complejo que comienza con el procesamiento de precursores que no poseen secuencias en común. La información estructural de alta resolución sobre proteínas involucradas en el proceso es escasa. La endoribonucleasa DCL1 es la enzima más importante involucrada en el proceso en plantas; sin embargo, no se conoce en detalle su funcionamiento. El estudio estructural y funcional de proteínas multidominio como DCL1 requiere la expresión por separado de cada dominio. Conocer el plegamiento de la proteína en estudio permitiría realizar elecciones racionales en la definición de los límites de los constructos a expresar. Por todo esto se planteó en este trabajo hacer un análisis estructural *in silico* de la región Plataforma-PAZ de DCL1, que es esencial en la determinación del tamaño del producto.

Para ello se comenzó generando una base de datos con secuencias de proteínas que tuvieran similitud con la homóloga humana DICER, la cual fue curada según parámetros de similitud. Luego se procedió a realizar una predicción de dominios, en base a los cuales se determinó una región de interés que fue extraída de cada una de las secuencias. Después de realizar un análisis de la identidad presente entre las mismas, se extrajo un subgrupo de secuencias el cual fue alineado. Se llevó a cabo la predicción de estructuras secundarias de algunas de las proteínas de interés y se comparó con la información presente en bibliografía. Se pudo observar que las predicciones tanto de dominios como de secuencias fueron muy buenas, correspondiéndose con lo esperado. Sin embargo el alineamiento no fue bueno, y esto puede deberse a la baja homología de secuencia presente entre las distintas proteínas. Finalmente se hizo un modelado estructural usando la información del alineamiento general y la estructura conocida de la homóloga humana.

Abstract

Analysis and structural modeling of the catalytic domains of DCL1 from *Arabidopsis thaliana*

Micro-RNAs (miRNAs) are a class of small RNAs that actively participate in fundamental biological processes such as the development of organisms and responses to environmental changes. Its biogenesis is a complex process that begins with the processing of precursors that do not have sequences in common. The high resolution structural information on proteins involved in the process is scarce. The endoribonuclease DCL1 is the most important enzyme involved in miRNA processing in plants; however, there is little experimental information on the enzyme structure and activity. Structural and functional studies of multidomain proteins such as DCL1 require the expression of isolated domain(s). Knowledge of the folding of the protein under study allows rational choices in the definition of the limits of the constructs to be expressed. In this work we propose to perform an in-silico structural analysis of the DCL1 Platform-PAZ region, essential in the determination of the small RNA product size.

With this purpose, a database was created with protein sequences that show similarity to the human homologue DICER. The database was cured according to similarity parameters. A prediction of domain architecture was performed, based on which a region of interest was determined. This region was extracted from each of the sequences. After performing an analysis of the identity between them, a subgroup of sequences was extracted and aligned. Prediction of secondary structures of some of the proteins of interest was carried out, and then compared with the information present in the literature. It was observed that the predictions of domains and sequences were very good, according to what was expected. However, the alignment was not good, probably due to the low sequence homology present between the different proteins. Finally, a structural model of the region of interest was obtained using the information of the general alignment and the known structure of the human homologue.

Índice

1. INTRODUCCIÓN

Los micro-ARNs (miARN) son una clase de ARNs pequeños endógenos que regulan negativamente la expresión de genes target, participando activamente en procesos biológicos fundamentales tales como el desarrollo de los organismos y las respuestas a cambios ambientales. Se considera que en animales aproximadamente un tercio de sus genes están regulados por estos mecanismos. Para cumplir su función es necesario que reconozcan secuencias complementarias en los ARNs mensajeros (Rana et al., 2007). Los miARNs están evolutivamente conservados y si bien hoy en día se encuentran anotados más de 28600 *(*miRBase, 2017), sólo se conocen las funciones biológicas de unos pocos.

La biogénesis de miARNs es un proceso complejo, en el cual el reconocimiento de estructura del precursor de ARN por las proteínas de procesamiento juega un rol clave para la identificación del producto miARN.(Bologna et al., 2009; Mateos et al., 2010; Werner et al., 2010; Song et al., 2010). El proceso comienza con la síntesis de los transcriptos primarios (pri-miARN) por la enzima ARN polimerasa II. En plantas, este pri-miARN es procesado en el núcleo por la ribonucleasa DCL1 (endoribonucleasa DICER-like 1) con la participación de las proteínas de unión a ARN doble hebra HYL1 y SERRATE (Xie et al., 2010). Luego este miARN es metilado y transportado al citoplasma donde cumple su función regulatoria (Figura 1).

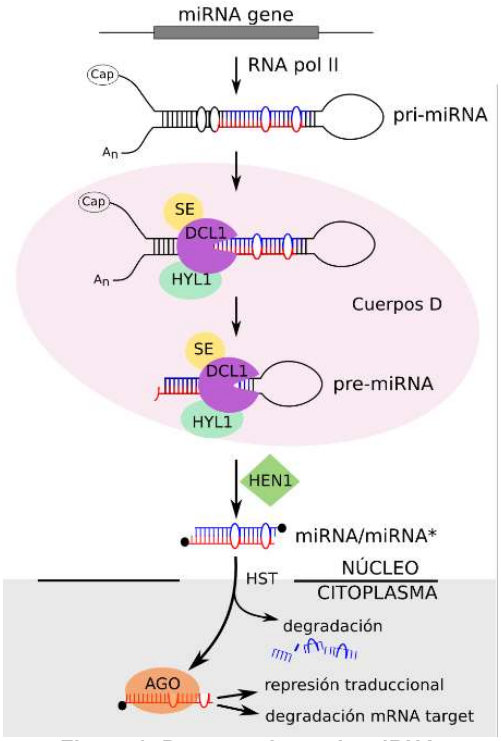


Figura 1: Procesamiento de miRNA

Los sustratos de DCL1 son oligonucleótidos de ARN cuya característica común es su estructura secundaria en forma de hebilla, con un tallo de doble hebra imperfecta que presenta desarreglos y protuberancias (“mismatches”, “bulges”). Si bien el complejo de procesamiento de plantas DCL1-HYL1-SERRATE es capaz de liberar con precisión la secuencia específica del miARN a partir del pri-miARN, el reconocimiento de los precursores es particularmente complejo debido a la heterogeneidad de los mismos (Xie et al., 2010), ya que no poseen una secuencia común que pueda ser reconocida ni se han podido identificar hasta el momento elementos de estructura que definan exactamente la posición del miARN en los mismos. El actual grado de conocimiento sobre los detalles mecanísticos y estructurales de estas proteínas no permite definir el modo de reconocimiento de sustrato o la especificidad en la reacción de corte del precursor.

La información estructural de alta resolución sobre proteínas que procesan los pri-miARN disponibles hasta el momento es escasa. Dentro del complejo de procesamiento las proteínas tipo-DICER resultan claves debido a que en ellas residen los dominios RNAsaIII responsables de la actividad catalítica. La proteína

DICER de *Giardia intestinalis* es la única que ha sido resuelta completa por cristalografía, (MacRae et al., 2006) pero carece de varios de los dominios característicos de las DICER de organismos más complejos. Por otra parte, se han resuelto fragmentos conteniendo el dominio RNAsaIII-2 y DRBM (dominios de unión a ARN doble hebra) de DICER de ratón (Du et al., 2008) y uno similar de una enzima tipo DICER de la levadura *Kluveromyces polysporus* (Weinberg et al., 2011) pero no constituyen estructuras de proteínas completas. En cuanto a la proteína DICER humana, es una enzima multidominio cuya mitad C-terminal incluye un dominio PAZ responsable del reconocimiento de los nucleótidos libres en el extremo 3' del ARN, un par de dominios RNaseIII en tándem que constituyen el centro activo, y un dominio de unión a ARN doble hebra (DRBM); mientras que en la parte N-terminal contiene un dominio Helicasa y un dominio DUF283 de función desconocida. Recientemente se obtuvo por cristalografía la estructura correspondiente al fragmento plataforma-PAZ de dicha proteína en complejo con un fragmento de ARNdh (Tian et al., 2014) en dos formas cristalinas diferentes, lo cual permitió sugerir la forma de los complejos de unión y de transferencia del precursor (en la parte superior de la Figura 2 se muestran los dominios de DICER, con una línea se indica la región cristalizada). Respecto a DCL1 de *Arabidopsis thaliana* solo se dispone de las estructuras en solución del segundo dominio DRBM resuelto por Resonancia Magnética Nuclear (Burdisso et al., 2012), por ello resulta interesante utilizar la información estructural conocida sobre otras especies para el análisis de la estructura de DCL1. En la parte inferior de la Figura 2 se muestra la distribución de los dominios para esta proteína.

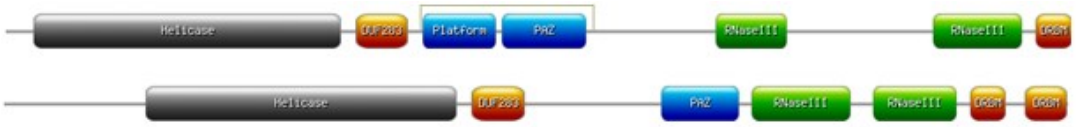


Figura 2: Arriba dominios de DICER humana, abajo dominios de DCL1 de *Arabidopsis thaliana*.

2. OBJETIVOS

El objetivo general es la obtención de un modelo estructural para la región plataforma-PAZ de DCL1 de *Arabidopsis thaliana* (Figura 3).

Figura 3: Dominios de DCL1, se muestra marcada la región de interés

Los objetivos particulares son:

- Generar una base de datos de secuencias de proteínas tipo DICER disponibles en el sitio de National Center of Biotechnology Information (NCBI)
- Realizar una identificación *ab initio* de los dominios de las secuencias seleccionadas y generar una nueva base de datos conteniendo sólo la región de interés
- Alinear las secuencias obtenidas
- Realizar una identificación *ab initio* de los elementos de estructura secundaria de proteínas seleccionadas
- Graficar en conjunto los resultados obtenidos para su análisis
- Realizar un alineamiento estructural de la región seleccionada de la proteína DCL1 de *Arabidopsis thaliana* con una parte de la proteína DICER de humano en complejo con ARNdh.

3. RESULTADOS

Este trabajo se presenta dentro del entorno de la aplicación web de ***Jupyter Notebook*** *(http://jupyter.org/). La misma cuenta con soporte para más de 40 lenguajes de programación, y permite trabajar de una forma interactiva, ordenada y prolija, intercalando comentarios y teoría con distintas líneas de código que se pueden correr de forma independiente.*

A su vez se decidió trabajar con el lenguaje Python por su librería de ***Biopython*** *que cuenta con múltiples herramientas para trabajar en biología (tutorial en http://biopython.org/DIST/docs/tutorial/Tutorial.html)*

3.1. Generar una base de datos

En esta primer parte se trabajó para generar una base de datos de secuencias de proteínas relacionadas tipo DICER, cuidando en todo momento de no perder de la lista las secuencias de DICER humana y de DCL1 de *Arabidopsis thaliana*.

En las primeras búsquedas se decidió trabajar con el buscador de NCBI accesible desde la página web *http://www.ncbi.nlm.nih.gov/*, el cual permite hacer distintas consultas y luego descargar los resultados obtenidos en el formato que sea de preferencia. Trabajando dentro de la opción "Protein" se probaron distintas *query* relacionadas a la base que se pretendía generar, tales como "DICER", "DCL1", "DICER-like". Al proceder de esta forma se obtenían como resultado cualquier registro que contuviese dichas palabras, por lo que muchos no eran los deseables ya sea por ser repetitivos, fragmentos cortos, o no corresponder a secuencias curadas. Se intentó filtrarlos según el largo de sus secuencias, o eliminar aquellos que incluían palabras como "hypothetical", pero no fue suficiente. La razón por la cual esta forma de búsqueda no era adecuada se relaciona con que el hecho de que el nombre que se le asigna a cada secuencia es subjetivo del usuario que la sube a la base de datos, sumado a que mucho de lo que se publica no se controla.

Para resolver este problema se decidió seleccionar proteínas a partir de su similitud de secuencia con respecto a alguna utilizada como base. Se trabajó con la herramienta BLAST (Basic Local Alignment Search Tool) que ofrece el mismo sitio en la página *http://blast.ncbi.nlm.nih.gov/Blast.cgi*, en la opción *Protein BLAST*, y se probaron distintos parámetros. Se seteó como Database a *UniProtKB/Swiss-Prot* que es una base curada con un alto nivel de anotación (como la descripción de la función de la proteína, la estructura de dominios, modificaciones post-traduccionales, variantes, etc.), un nivel de redundancia mínimo y un alto grado de integración con otras base de datos (*http://www.expasy.org/sprot* , *http://www.uniprot.org/help/about*).

Durante gran parte del tiempo se trabajó con un BLAST realizado sobre la secuencia completa de DCL1 (gi -> 15223286, que se reconoció automáticamente como `NP_171612:dicer-like 1 [Arabidopsis thaliana]`). Esto dio como resultado un gran número de Hits (>100), por lo que se decidió conservar los primeras 100, y luego filtrar el resultado conservando solo aquellos casos que en su nombre incluyesen la palabra "DICER", reduciendo así la cantidad a 44. Cuando se prosiguió con el resto del trabajo se encontró que dicha base contenía secuencias que eran demasiado divergentes con respecto a la original en la zona que era de interés, con lo cual se complicaba el análisis. La razón más probable por la cual ocurría esto es el haber trabajado con la secuencia completa.

Finalmente se probó y decidió realizar el BLAST sobre la secuencia de interés, es decir, la de la estructura cristalográfica del complejo de la porción de DICER humana con RNA. Los parámetros seleccionados fueron:

```
Query sequence ->
SDQPCYLYVIGMVLTTPLPDELNFRRRKLYPPEDTTRCFGILTAKPIPQIPHFPVYT
RSGEVTISIELAASGFMLSLQMLELITRLHQYIFSHILRLEKPALEFKPTDADSAYC
VLPLNVVNDSSTLDIDFKFMEDIEKSEARIGIPSTKYTKETPFVFKLEDYQDAVIIP
RYRNFDQPHRFYVADVYTDLTPLSKFPSPEYETFAEYYKTKYNLDLTNLNQPLLDVD
HTSSRLNLLTPRHLNQKGKALPLSSAEKRKAKWESLQNKQILVPELCAIHPIPASLW
RKAVCLPSILYRLHCLL
Database -> swissprot
Matrix -> PAM70
```

El resto de los parámetros fueron los default. En la Figura 4 se puede observar la salida que se obtuvo con dicha búsqueda:

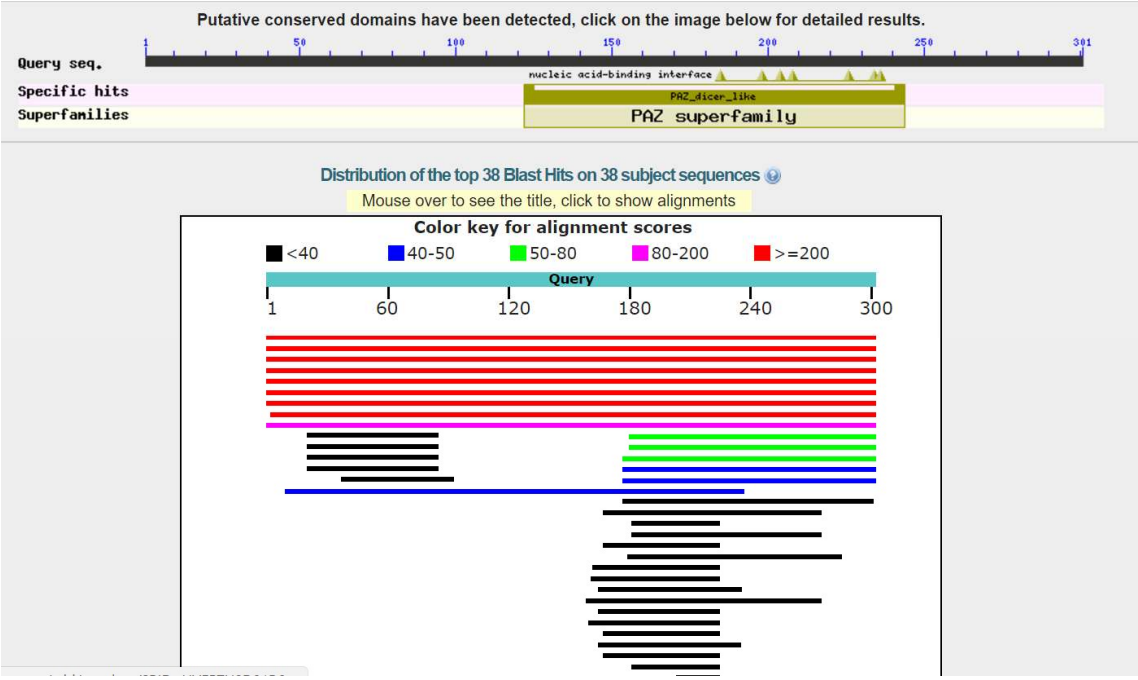


Figura 4: Búsqueda BLAST

Para evitar trabajar con una base de datos estática que con el tiempo estuviese desactualizada se decidió generar un sistema que permita realizar la búsqueda en cualquier momento y tener resultados completos al día. Dentro del paquete de *Biopython* existen herramientas para hacer esto y que permiten guardar el resultado en una variable (incluyendo parámetros de la búsqueda, alineamientos, etc.). Es necesario setear algunas variables principales: el programa que se desea utilizar, la base de datos sobre la que se va a realizar la búsqueda y la secuencia query. Se agregó un parámetro opcional relacionado con la matriz que era de interés, y otro que extiende el número de resultados de 50 (default) a 100.

```
In [1]:
#Defino la secuencia (parte de DICER humana que formó complejo
cristalográfico)
seq_4ngd = "\
SDQPCYLYVIGMVLTTPLPDELNFRRRKLYPPEDTTRCFGILTAKPIPQIPHFPVYTRSGE\
VTISIELAASGFMLSLQMLELITRLHQYIFSHILRLEKPALEFKPTDADSAYCVLPLNVVN\
DSSTLDIDFKFMEDIEKSEARIGIPSTKYTKETPFVFKLEDYQDAVIIPRYRNFDQPHRFY\
VADVYTDLTPLSKFPSPEYETFAEYYKTKYNLDLTNLNQPLLDVDHTSSRLNLLTPRHLNQ\
KGKALPLSSAEKRKAKWESLQNKQILVPELCAIHPIPASLWRKAVCLPSILYRLHCLL"

from Bio.Blast import NCBIWWW
#help(NCBIWWW.qblast) #Para conocer todas las opciones posibles se
puede correr
res_BLAST = NCBIWWW.qblast(program="blastp", database="swissprot",
sequence= seq_4ngd, matrix_name="PAM70", hitlist_size="100")

#Parseo del resultado
from Bio.Blast import NCBIXML
res_BLAST_parsed = NCBIXML.read(res_BLAST)
base1=list(res_BLAST_parsed.alignments)

#Número de secuencias alineadas
print "El número de secuencias es ", len(base1)
```

```
El número de secuencias es  44
```

Como resultado del BLAST sobre la secuencia *4ngd* se pudieron obtener 44 hits que se guardaron dentro de la variable **base1**. El título de cada secuencia esta compuesto por 5 partes separadas por "|": gi (identificador de secuencia), valor de gi (serie de dígitos que se asignan consecutivamente a cada registro de secuencia procesada por NCBI), sp, Sequence ID, y nombre de la proteína. Se decidió extraer para cada secuencia su nombre, largo, score y e-value para tener una idea general de la forma y calidad del resultado obtenido. Se graficó los valores de E-value obtenidos para todas las secuencias.

- *score* es un valor numérico que describe la calidad general del alineamiento (mayor *score*, mayor similaridad). La escala depende del sistema usado, la matriz de sustitución y la penalidad de los gaps.
- *E-value* (Expectation value) asociado a un Score S es un parámetro que describe el número de hits que se esperaría que uno pueda ver al azar

cuando se busca en una base de datos de un tamaño en particular. Su valor decrece exponencialmente a medida que el valor de Score (S) del alineamiento se incrementa. Por ejemplo, un valor de E-value=1 asignado a un dado hit significa que en una base de datos de ese tamaño se espera encontrar un acierto con un score similar solo por casualidad (menor E-value significa que el Score es más significante). El alineamiento de BLAST está ordenado según este parámetro. Es un parámetro conveniente para establecer un umbral de significación a la hora de reportar los resultados, y se prefiere por encima del Score porque a diferencia del último es menos sensible al largo de la secuencia.

E-value(S)=P-value(S)x(largo de la secuencia query)x(largo de la base de datos)

```
In [2]:
i = 0
nro_base1 = []
name_base1 = []
evalue_base1 = []

for seq in base1[:]:
    nro_base1.append(i)
    name_base1.append(seq.title.split('|')[-1].split()[0])
    evalue_base1.append(seq.hsps[0].expect)
    print (str(seq.title.split('|')[-1]) + "\nlength: "
           + str(seq.length) + "\t\tscore: " \
           + str(seq.hsps[0].score) + "\t\tevalue: "
           + str(seq.hsps[0].expect) + "\n-----------------\n")
    i = i + 1
```

```
DICER_HUMAN RecName: Full=Endoribonuclease Dicer; AltName:
Full=Helicase with RNase motif; Short=Helicase MOI
length: 1922        score: 1485.0       evalue: 0.0
-------------------
DICER_BOVIN RecName: Full=Endoribonuclease Dicer
length: 1923        score: 1473.0       evalue: 0.0
-------------------
DICER_MOUSE RecName: Full=Endoribonuclease Dicer; AltName:
Full=Double-strand-specific ribonuclease mDCR-1
length: 1916        score: 1465.0       evalue: 0.0
-------------------
DICER_CRIGR RecName: Full=Endoribonuclease Dicer
length: 1917        score: 1460.0       evalue: 0.0
-------------------
DICER_CHICK RecName: Full=Endoribonuclease Dicer
length: 1921        score: 1456.0       evalue: 0.0
-------------------
DICER_DANRE RecName: Full=Endoribonuclease Dicer
length: 1865        score: 1414.0       evalue: 0.0
-------------------
DICER_XENTR RecName: Full=Endoribonuclease Dicer
length: 1893        score: 1397.0       evalue: 0.0
-------------------
DCR1_DROME RecName: Full=Endoribonuclease Dcr-1; Short=Protein
dicer-1
length: 2249        score: 681.0        evalue: 5.53592e-84
-------------------
DCR1_CAEEL RecName: Full=Endoribonuclease dcr-1
length: 1910        score: 461.0        evalue: 5.97677e-53
-------------------
DCL1_ARATH RecName: Full=Endoribonuclease Dicer homolog 1;
AltName: Full=Dicer-like protein 1; Short=AtDCL1; AltName:
Full=Protein ABNORMAL SUSPENSOR 1; AltName: Full=Protein
CARPEL FACTORY; AltName: Full=Protein SHORT INTEGUMENTS 1;
AltName: Full=Protein SUSPENSOR 1
length: 1909        score: 168.0        evalue: 2.46213e-13
-------------------
DCL1_ORYSJ RecName: Full=Endoribonuclease Dicer homolog 1;
AltName: Full=Dicer-like protein 1; Short=OsDCL1
length: 1883        score: 152.0        evalue: 2.88829e-11
-------------------
DCL3_ARATH RecName: Full=Endoribonuclease Dicer homolog 3;
```

```
AltName: Full=Dicer-like protein 3; Short=AtDCL3
length: 1580        score: 109.0        evalue: 1.84441e-05
-------------------
DCL2B_ORYSJ RecName: Full=Endoribonuclease Dicer homolog 2b;
AltName: Full=Dicer-like protein 2b; Short=OsDCL2b
length: 1377        score: 103.0        evalue: 0.000120872
-------------------
DCL2A_ORYSJ RecName: Full=Endoribonuclease Dicer homolog 2a;
AltName: Full=Dicer-like protein 2a; Short=OsDCL2a
length: 1410        score: 102.0        evalue: 0.0001727
-------------------
DCL3A_ORYSJ RecName: Full=Endoribonuclease Dicer homolog 3a;
AltName: Full=Dicer-like protein 3a; Short=OsDCL3a
length: 1651        score: 85.0     evalue: 0.0306418
-------------------
DCL1_MAGO7 RecName: Full=Dicer-like protein 1; Includes:
RecName: Full=Endoribonuclease DCL1; Includes: RecName:
Full=ATP-dependent helicase DCL1
length: 1591        score: 81.0     evalue: 0.0951829
-------------------
DCL4_ORYSJ RecName: Full=Endoribonuclease Dicer homolog 4;
AltName: Full=Dicer-like protein 4; Short=OsDCL4; AltName:
Full=Protein SHOOT ORGANIZATION 1
length: 1657        score: 81.0     evalue: 0.0980699
-------------------
AUB_DROME RecName: Full=Protein aubergine; AltName:
Full=Protein sting
length: 866     score: 81.0     evalue: 0.108596
-------------------
PIWL1_MOUSE RecName: Full=Piwi-like protein 1
length: 862     score: 80.0     evalue: 0.155009
-------------------
PIWL1_HUMAN RecName: Full=Piwi-like protein 1
length: 861     score: 79.0     evalue: 0.165801
-------------------
PIWL1_DANRE RecName: Full=Piwi-like protein 1
length: 858     score: 76.0     evalue: 0.425688
-------------------
PIWL2_DANRE RecName: Full=Piwi-like protein 2
length: 1046        score: 75.0     evalue: 0.556491
-------------------
PIWL1_CHICK RecName: Full=Piwi-like protein 1
length: 867     score: 75.0     evalue: 0.644168
```

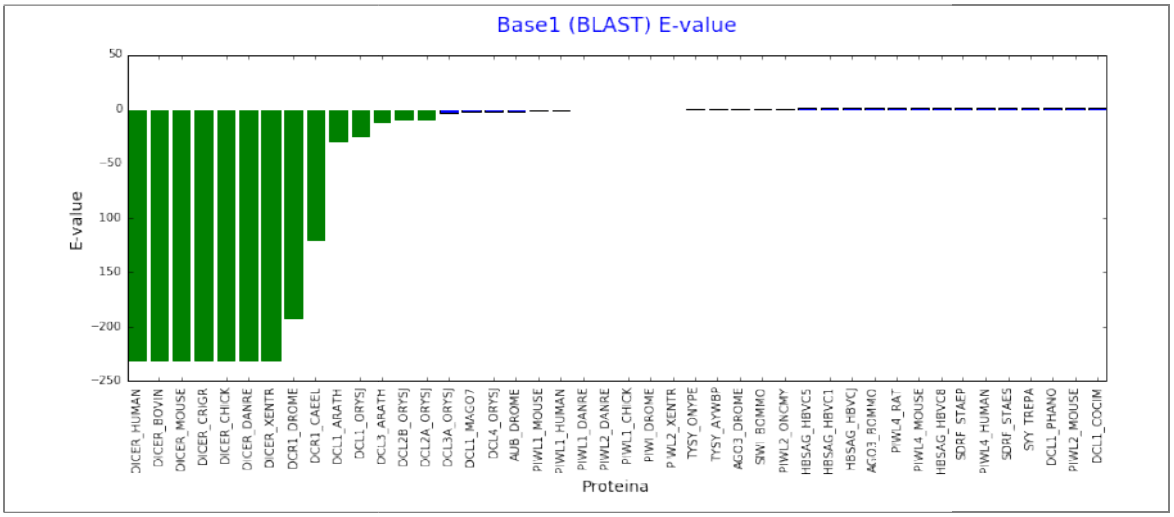
```
-------------------
PIWI_DROME RecName: Full=Protein piwi
length: 843     score: 75.0     evalue: 0.649572
-------------------
PIWL2_XENTR RecName: Full=Piwi-like protein 2
length: 949     score: 74.0     evalue: 0.791424
-------------------
TYSY_ONYPE RecName: Full=Thymidylate synthase; Short=TS;
Short=TSase
length: 288     score: 73.0     evalue: 1.12123
-------------------
TYSY_AYWBP RecName: Full=Thymidylate synthase; Short=TS;
Short=TSase
length: 288     score: 73.0     evalue: 1.15424
-------------------
AGO3_DROME RecName: Full=Protein argonaute-3
length: 867     score: 70.0     evalue: 2.97467
-------------------
SIWI_BOMMO RecName: Full=Piwi-like protein Siwi
length: 899     score: 70.0     evalue: 2.979
-------------------
PIWL2_ONCMY RecName: Full=Piwi-like protein 2
length: 1054        score: 70.0     evalue: 3.02524
-------------------
HBSAG_HBVC5 RecName: Full=Large envelope protein; AltName:
Full=L glycoprotein; AltName: Full=L-HBsAg; Short=LHB;
AltName: Full=Large S protein; AltName: Full=Large surface
protein; AltName: Full=Major surface antigen
length: 400     score: 69.0     evalue: 3.24206
-------------------
HBSAG_HBVC1 RecName: Full=Large envelope protein; AltName:
Full=L glycoprotein; AltName: Full=L-HBsAg; Short=LHB;
AltName: Full=Large S protein; AltName: Full=Large surface
protein; AltName: Full=Major surface antigen
length: 400     score: 69.0     evalue: 3.4023
-------------------
HBSAG_HBVCJ RecName: Full=Large envelope protein; AltName:
Full=L glycoprotein; AltName: Full=L-HBsAg; Short=LHB;
AltName: Full=Large S protein; AltName: Full=Large surface
protein; AltName: Full=Major surface antigen
length: 400     score: 69.0     evalue: 3.67534
-------------------
AGO3_BOMMO RecName: Full=Piwi-like protein Ago3; Short=BmAGO3
```

```
length: 926     score: 69.0     evalue: 4.29868
-------------------
PIWL4_RAT RecName: Full=Piwi-like protein 4
length: 848     score: 69.0     evalue: 4.40933
-------------------
PIWL4_MOUSE RecName: Full=Piwi-like protein 4; Short=mAgo5
length: 848     score: 69.0     evalue: 4.40933
-------------------
HBSAG_HBVC8 RecName: Full=Large envelope protein; AltName:
Full=L glycoprotein; AltName: Full=L-HBsAg; Short=LHB;
AltName: Full=Large S protein; AltName: Full=Large surface
protein; AltName: Full=Major surface antigen
length: 389     score: 68.0     evalue: 5.49788
-------------------
SDRF_STAEP RecName: Full=Serine-aspartate repeat-containing
protein F; Flags: Precursor
length: 1733        score: 68.0     evalue: 5.66944
-------------------
PIWL4_HUMAN RecName: Full=Piwi-like protein 4
length: 852     score: 68.0     evalue: 5.71849
-------------------
SDRF_STAES RecName: Full=Serine-aspartate repeat-containing
protein F; Flags: Precursor
length: 1633        score: 68.0     evalue: 5.82836
-------------------
SYY_TREPA RecName: Full=Tyrosine--tRNA ligase; AltName:
Full=Tyrosyl-tRNA synthetase; Short=TyrRS
length: 409     score: 67.0     evalue: 6.20055
-------------------
DCL1_PHANO RecName: Full=Dicer-like protein 1; Includes:
RecName: Full=Endoribonuclease DCL1; Includes: RecName:
Full=ATP-dependent helicase DCL1
length: 1522        score: 67.0     evalue: 7.40039
-------------------
PIWL2_MOUSE RecName: Full=Piwi-like protein 2
length: 971     score: 67.0     evalue: 8.04684
-------------------
DCL1_COCIM RecName: Full=Dicer-like protein 1; Includes:
RecName: Full=Endoribonuclease DCL1; Includes: RecName:
Full=ATP-dependent helicase DCL1
length: 1500        score: 66.0     evalue: 8.88035
-------------------
```

```
In [3]:
#Gráficos Prteínas vs E-value
import matplotlib.pyplot as plt
%matplotlib inline
from numpy import *

evalue_base1_sincero = [(1e-100) if x==0 else x for x in
evalue_base1]
plt.rcParams["figure.figsize"] = [15,5]
plt.xticks(range(0,44), name_base1, rotation=90)
barlist= plt.bar(nro_base1, log(evalue_base1_sincero),
align='center')
plt.title("Base1 (BLAST) E-value", size=18, y = 1.05, color = "b")
plt.xlabel("Proteina", size = 14)
plt.ylabel('E-value', size = 14)

for n,v in enumerate(evalue_base1):
    if v<0.01:
        barlist[n].set_color("g")
```



En base a los resultados obtenidos y usando como referencia valores encontrados en bibliografía, se decidió poner un umbral al valor de E-value de 0.01, es decir, conservar en la base de datos todas las proteínas para las que su valor de E-value haya sido igual o menor a 0.01. Se generó así la **base2**.

```
In [4]:
#Subgrupo con e-value menor a 0.01
base2=[]
for seq in base1[:]:
    if seq.hsps[0].expect < 0.01:
        base2.append(seq)

#Visualización de la base2
i = 1
for seq in base2:
    print i, seq.title.split('|')[-1][:11]
    i = i + 1
```

```
1 DICER_HUMAN
2 DICER_BOVIN
3 DICER_MOUSE
4 DICER_CRIGR
5 DICER_CHICK
6 DICER_DANRE
7 DICER_XENTR
8 DCR1_DROME
9 DCR1_CAEEL
10 DCL1_ARATH
11 DCL1_ORYSJ
12 DCL3_ARATH
13 DCL2B_ORYSJ
14 DCL2A_ORYSJ
```

Esta nueva base contiene 14 secuencias de proteínas, y respeta contener a DICER humana y a DCL1 de *Arabidopsis thaliana*. A continuación se decidió recuperar las secuencias completas de todas ellas, y guardarlas en un archivo de formato tipo FASTA, el cual permite luego ser utilizado por los programas de alineamiento y predicción. Para la obtención de las secuencias se recurrió a la base ExPASy (*https://www.expasy.org/*), buscando a partir de los IDs conocidos y siguiendo el uso de los paquetes de Biopython.

Luego de hacer la búsqueda no se pudo obtener la secuencia para una de las proteínas (DCR1_CAEEL) por alguna divergencia en la forma en que estaba anotada; sin embargo no era una por la que se tenía interés en particular así que se eliminó de la lista. Luego se agregó externamente la variante del organismo *Giardia*

para incluirla en los análisis posteriores, y el fragmento *4ngd*, quedando un total de 15 secuencias en lo que se definió como **base3**. Se utilizó SeqIO que es una interfaz estándar de secuencia Input/Output para escribir un archivo en formato fasta llamado *"base3.fasta"* con la información del array.

```
In [5]:
#Secuencias completas de las proteínas del subgrupo anterior
from Bio import ExPASy
from Bio import SwissProt
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord

base2_IDs_descs_seq=[]

for d in base2:
    handle = ExPASy.get_sprot_raw(d.title.split("|")[3])
    try:
        record = SwissProt.read(handle)
        base2_IDs_descs_seq.append(SeqRecord(Seq(record.sequence),
        id="%s"%(d.title.split("|")[3]),
        description=d.title.split("|")[-1].split()[0]))
    except:
        print  "No se pudo obtener la secuencia para", d.title


#Guardar en fasta
from Bio import SeqIO
SeqIO.write(base2_IDs_descs_seq,"base3.fasta","fasta")

fasta_Giardia="\
>A8BQJ3
DCL_GIAIC\nMHALGHCCTVVTTRGPSHWLLLLDTHLGTLPGFKVSAGRGLPAAEVYFEAGPRVS\
LSRTD\nATIVAVYQSILFQLLGPTFPASWTEIGATMPHNEYTFPRFISNPPQFATLAFLPLLSPT\
S\nPLDLRALMVTAQLMCDAKRLSDEYTDYSTLSASLHGRMVATPEISWSLYVVLGIDSTQTS\nL\
SYFTRANESITYMRYYATAHNIHLRAADLPLVAAVRLDDLKDHQIPAPGSWDDLAPKLR\nFLPPE\
LCLLLPDEFDLIRVQALQFLPEIAKHICDIQNTICALDKSFPDCGRIGGERYFAI\nTAGLRLDQG\
RGRGLAGWRTPFGPFGVSHTDVFQRLELLGDAVLGFIVTARLLCLFPDASV\nGTLVELKMELVRN\
EALNYLVQTLGLPQLAEFSNNLVAKSKTWADMYEEIVGSIFTGPNGI\nYGCEEFLAKTLMSPEHS\
KTVGSACPDAVTKASKRVCMGEAGAHEFRSLVDYACEQGISVF\nCSSRVSTMFLERLRDIPAEDM\
LDWYRLGIQFSHRSGLSGPGGVVSVIDIMTHLARGLWLG\nSPGFYVEQQTDKNESACPPTIPVLY\
IYHRSVQCPVLYGSLTETPTGPVASKVLALYEKIL\nAYESSGGSKHIAAQTVSRSLAVPIPSGTI\
PFLIRLLQIALTPHVYQKLELLGDAFLKCSL\nALHLHALHPTLTEGALTRMRQSAETNSVLGRLT\
KRFPSVVSEVIIESHPKIQPDSKVYGD\nTFEAILAAILLACGEEAAGAFVREHVLPQVVADA\n"
```

```
fasta_4ngd = "\
>4ngd\nSDQPCYLYVIGMVLTTPLPDELNFRRRKLYPPEDTTRCFGILTAKPIPQIPHFPVYTRS\
G\nEVTISIELAASGFMLSLQMLELITRLHQYIFSHILRLEKPALEFKPTDADSAYCVLPLNV\nV\
NDSSTLDIDFKFMEDIEKSEARIGIPSTKYTKETPFVFKLEDYQDAVIIPRYRNFDQPH\nRFYVA\
DVYTDLTPLSKFPSPEYETFAEYYKTKYNLDLTNLNQPLLDVDHTSSRLNLLTPR\nHLNQKGKAL\
PLSSAEKRKAKWESLQNKQILVPELCAIHPIPASLWRKAVCLPSILYRLHC\nLL\n"

archi=open("base3.fasta", "a")
archi.write(fasta_Giardia)
archi.write(fasta_4ngd)
archi.close()
```

```
No se pudo obtener la secuencia para
gi|408360292|sp|P34529.3|DCR1_CAEEL RecName:
Full=Endoribonuclease dcr-1
```

La nueva base quedó conformada por las siguientes proteínas (se detallan además los nombres científicos de los organismos a los cuales pertenece cada una de ellas):

- 4ngd *Homo sapiens* (Human)
- DCL_GIAIC *Giardia intestinalis* (Giardia lamblia)
- DCL2A_ORYSJ *Oryza sativa* subsp. japonica (Rice)
- DCL2B_ORYSJ *Oryza sativa* subsp. japonica (Rice)
- DCL3_ARATH *Arabidopsis thaliana* (Mouse-ear cress)
- DCL1_ORYSJ *Oryza sativa* subsp. japonica (Rice)
- DCL1_ARATH *Arabidopsis thaliana* (Mouse-ear cress)
- DCR1_DROME *Drosophila melanogaster* (Fruit fly)
- DICER_XENTR *Xenopus tropicalis* (Western clawed frog)
- DICER_DANRE *Danio rerio* (Zebrafish)
- DICER_CHICK *Gallus gallus* (Chicken)
- DICER_CRIGR *Cricetulus griseus* (Chinese hamster)
- DICER_MOUSE *Mus musculus* (Mouse)
- DICER_BOVIN *Bos taurus* (Bovine)
- DICER_HUMAN *Homo sapiens* (Human).

```
In [6]:
#Visualización de la primer parte base3.fasta
f = open('base3.fasta', 'r')
file_contents = f.read()
print (file_contents [0:2200])
f.close()
```

```
>Q9UPY3.3 DICER_HUMAN
MKSPALQPLSMAGLQLMTPASSPMGPFFGLPWQQEAIHDNIYTPRKYQVELLEAALDHNT
IVCLNTGSGKTFIASTTLLKSCLYLDLGETSARNGKRTVFLVNSANQVAQQVSAVRTHSD
LKVGEYSNLEVNASWTKERWNQEFTKHQVLIMTCYVALNVLKNGYLSLSDINLLVFDECH
LAILDHPYREFMKLCEICPSCPRILGLTASILNGKWDPEDLEEKFQKLEKILKSNAETAT
DLVVLDRYTSQPCEIVVDCGPFTDRSGLYERLLMELEEALNFINDCNISVHSKERDSTLI
SKQILSDCRAVLVVLGPWCADKVAGMMVRELQKYIKHEQEELHRKFLLFTDTFLRKIHAL
CEEHFSPASLDLKFVTPKVIKLLEILRKYKPYERHSFESVEWYNNRNQDNYVSWSDSEDD
DEDEEIEEKEKPETNFPSPFTNILCGIIFVERRYTAVVLNRLIKEAGKQDPELAYISSNF
ITGHGIGKNQPRNNTMEAEFRKQEEVLRKFRAHETNLLIATSIVEEGVDIPKCNLVVRFD
LPTEYRSYVQSKGRARAPISNYIMLADTDKIKSFEEDLKTYKAIEKILRNKCSKSVDTGE
TDIDPVMDDDHVFPPYVLRPDDGGPRVTINTAIGHINRYCARLPSDPFTHLAPKCRTREL
PDGTFYSTLYLPINSPLRASIVGPPMSCVRLAERVVALICCEKLHKIGELDDHLMPVGKE
TVKYEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRDSYPRPDQPCYLYVIGMVLT
TPLPDELNFRRRKLYPPEDTTRCFGILTAKPIPQIPHFPVYTRSGEVTISIELKKSGFML
SLQMLELITRLHQYIFSHILRLEKPALEFKPTDADSAYCVLPLNVVNDSSTLDIDFKFME
DIEKSEARIGIPSTKYTKETPFVFKLEDYQDAVIIPRYRNFDQPHRFYVADVYTDLTPLS
KFPSPEYETFAEYYKTKYNLDLTNLNQPLLDVDHTSSRLNLLTPRHLNQKGKALPLSSAE
KRKAKWESLQNKQILVPELCAIHPIPASLWRKAVCLPSILYRLHCLLTAEELRAQTASDA
GVGVRSLPADFRYPNLDFGWKKSIDSKSFISISNSSSAENDNYCKHSTIVPENAAHQGAN
RTSSLENHDQMSVNCRTLLSESPGKLHVEVSADLTAINGLSYNQNLANGSYDLANRDFCQ
GNQLNYYKQEIPVQPTTSYSIQNLYSYENQPQPSDECTLLSNKYLDGNANKSTSDGSPVM
AVMPGTTDTIQVLKGRMDSEQSPSIGYSSRTLGPNPGLILQALTLSNASDGFNLERLEML
GDSFLKHAITTYLFCTYPDAHEGRLSYMRSKKVSNCNLYRLGKKKGLPSRMVVSIFDPPV
NWLPPGYVVNQDKSNTDKWEKDEMTKDCMLANGKLDEDYEEEDEEEESLMWRAPKEEADY
EDDFLEYDQEHIRFIDNMLMGSGAFVKKISLSPFSTTDSAYEWKMPKKSSLGSMPFSSDF
EDFDYSSWDAMCYLDPSKAVEEDDFVVGFWNPSEENCGVDTGKQSISYDLHTEQCIADKS
IADCVEALLGCYLTSCGERAAQLFLCSLGLKVLPVIKRTDREKALCPTRENFNSQQKNLS
VSCAAASVASSRSSVLKDSEYGCLKIPPRCMFDHPDADKTLNHLISGFENFEKKINYRFK
NKAYLLQAFTHASYHYNTITDCYQRLEFLGDAILDYLITKHLYEDPRQHSPGVLTDLRSA
LVNNTIFASLAVKYDYHKYFKAVSPELFHVIDDFVQFQLEKNEMQGMDSELRRSEEDEEK
EEDIEVPKAMGDIFESLAGAIYMDSGMSLETVWQVYYPMMRPLIEKFSANVPRSPVRELL
EMEPETAKFSPAERTYDGKVRVTVEVVGKGKFKGVGRSYRIAKSAAARRALRSLKANQPQ
VPNS
>Q6TUI4.3 DICER_BOVIN
MKSPALQPLSMAGLQLMTPASSPMGPFFGLPWQQEAIHDNIYTPRKYQVELLEAALDHNT
```

```
IVCLNTGSGKTFIAVLLTKELSYQIRGDFNRNGKRTVFLVNSANQVAQQVSAVRTHSDLK
VGEYSNLEVSASWTKEKWNQEFTKHQVLIMTCYVALNVLKNGYLSLSDINLLVFDECHLA
ILDHPYREIMKLCENC
```

En un primer momento se había decidido pasar directamente a la etapa de alineamiento, para luego continuar con las predicciones de estructura como se había propuesto en el proyecto original. Sin embargo, tanto cuando se estaba trabajando con resultados de BLAST sobre DCL1 completa, como con la nueva base en función de *4ngd*, los alineamientos no eran buenos en la zona de interés por estar trabajando con secuencias completas que al contener regiones divergentes complicaban el alineamiento. Por ello se resolvió posteriormente que, una vez armada la base de datos, era mejor pasar directamente a la predicción de dominios para luego en base a su resultado conservar solo las porciones de interés de cada proteína antes de decidir alinearlas.

3.2. Predicción de dominios y generación de una nueva base de datos

Para la predicción de dominios se utilizó el programa **ScanProsite** (*http://prosite.expasy.org/scanprosite/*), que tiene un tutorial dentro del de Biopython en la sección 10.6 (*http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc146*). En la Figura 5 se muestra parte del programa.

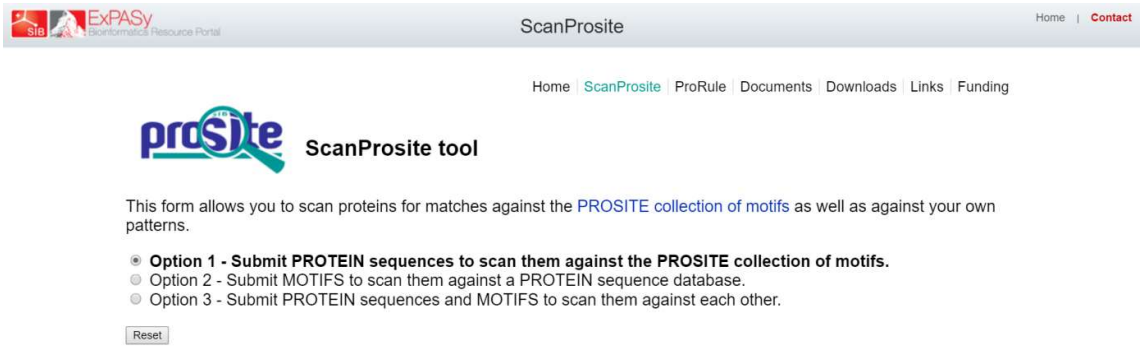


Figura 5: Programa Prosite

Una vez que el programa corre sobre alguna de las secuencias, devuelve una respuesta del tipo "Bio.ExPASy.ScanProsite.Record" donde se dan detalles de cada dominio encontrado, como su ID y posición. Sin embargo, es necesario que dichos IDs sean traducidos a nombres de dominios conocidos para poder interpretarlos. Para ello se descargó el *archivo prorule.dat.txt* desde el sitio ftp://ftp.expasy.org/databases/prosite/, en el cual se presenta a continuación de cada ID el nombre del dominio separado por comillas. Se definió una función llamada *id_in_rules*, que busca la posición en la cual se encuentra el ID dentro del archivo, y posteriormente identifica las comillas que contienen el nombre del dominio para extraerlo y guardarlo en la lista *name_dom*.

Se hizo la predicción de dominios para cada secuencia de la lista, luego se aplicó la función creada anteriormente para traducir los ID de dominios encontrados a nombres, y se los fue agregando a una lista llamada id_domain, la cual va a contener en última instancia todos los dominios encontrados en las secuencias del

archivo. Por otra parte se creó para cada secuencia un string formado por una repetición de símbolos "-" del largo de la misma. Finalmente se tomó las posiciones de comienzo y final de cada dominio, y se reemplazó en los rangos correspondientes dentro de la secuencia de símbolos "-" con el índice que se asignó a cada dominio y se guardó en una lista.

Trabajando de esta manera las predicciones de dominios quedaron representadas como secuencias de caracteres del largo de cada proteína, en la que en cada posición se ubicó un signo "-" en los casos donde nos hubo predicción, o una letra dada en los casos donde se encontró un dominio, respetando siempre el uso de la misma letra para el mismo dominio durante toda la base del datos. Por ej., una secuencia de 10 residuos en donde los primeros 5 hayan sido predichos como un dominio representado por una letra "a" se verá como:

aaaaa-----

Los resultados de la predicción se guardaron en un archivo fasta llamado *base3_dominio.fasta* y se graficaron para su análisis

```
In [7]:
#Parseo de la base3
from Bio.SeqIO import *

name_base3 = []
for rec in parse('base3.fasta', "fasta"):
    name_base3.append(str(rec.description))

seq_base3 = []
for rec in parse('base3.fasta', "fasta"):
    seq_base3.append(str(rec.seq))
```

```
In [8]:
#Definiendo funcion para cambiar id por nombres de dominios
def id_in_rules(i):
    file_rules = open("prorule.dat.txt", "r")
    text_rules = file_rules.read()
    if text_rules.find(i)>1:
        position_id = text_rules.find(i)
        first_dotcoma = text_rules.find(";", position_id)
        second_dotcoma = text_rules.find(";", (first_dotcoma + 1))
        name_dom = text_rules[(first_dotcoma + 2):second_dotcoma]
```

```
        file_rules.close()
        return name_dom

#Predicción de dominios
from Bio.ExPASy import ScanProsite
id_domain = {}

n = 97
domain_base3 = []

for seq in seq_base3:
    handle = ScanProsite.scan(seq)
    response = ScanProsite.read(handle)
#    print response #Descomentar si se quisiese ver la salida para
     cada predicción

     #Definición de una secuencia compuesto solo de "-" del largo de
      seq
    list_seq_domain = list("-" * len(seq))

    #Iteración en cada dominio encontrado en esa seq
    for item in response:
        iddom = item["signature_ac"]
        namedom = id_in_rules(iddom)

        #Creación de la variable idnum que va a servir para numerar
         los dominios nuevos
        idnum = -1

        #Chequeo de la presencia del nombre del dominio en la lista,
       #sino está se agrega con el índice idnum = n
        if namedom not in id_domain.values():
            id_domain[n] = namedom
            idnum = n
            n = (n + 1)

        #Si ya estaba en la lista se recupera con el indice que
     tenía
        else:
            for num, name in id_domain.iteritems():
                if name == namedom:
                    idnum = num
                    break

        #Busqueda de las posiciones de comienzo y final del dominio
         en la secuencia
        iddom_start = item["start"]
```

```
        iddom_stop = item["stop"]

        #Reemplazo en la secuencia de dichos rangos por idnum
        for i in range(iddom_start-1, iddom_stop):
            list_seq_domain[i] = chr(idnum)

    #Unión en la lista para formar un string que se agrega a la
      lista de la base
    domain_base3.append("".join(list_seq_domain))


#Definiendo los nombres de los dominios para cada letra asignada
st_id_domain=""
for key,val in id_domain.iteritems():
    st_id_domain = st_id_domain + chr(key) + "-> "+ str(val) + "   "
print st_id_domain

print "El número de predicciones de dominios es", len(domain_base3)
```

```
a-> HELICASE_ATP_BIND_1   b-> HELICASE_CTER   c-> DICER_DSRBF
d-> PAZ   e-> RNASE_3_2   f-> RNASE_3_1   g-> DS_RBD   h->
None
El número de predicciones de dominios es 15
```

```
In [9]:
#Guardado en archivo
base3_dominios_file = open("base3_dominio.fasta", "a")
for i, seq in enumerate(seq_base3):
    base3_dominios_file.write(">")
    base3_dominios_file.write(name_base3[i])
    base3_dominios_file.write("\n")
    base3_dominios_file.write(domain_base3[i])
    base3_dominios_file.write("\n")
base3_dominios_file.close()
```

```
In [10]:
#Visualización de la primer parte base3.fasta
f = open('base3_dominio.fasta', 'r')
file_contents = f.read()
print (file_contents [0:2200])
f.close()
```

```
>Q9UPY3.3 DICER_HUMAN
-------------------------------------------------
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa-------
------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
------------
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb----------------
-----------
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccc-------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
-------------
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddd----------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
--------------------
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeee---------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
-----------
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeffffffffeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
ggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggg
gggg--------
>Q6TUI4.3 DICER_BOVIN
-------------------------------------------------
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa----
```

```
In [11]:
#Para cargar nuevamente la variable sin tener que correr la
predicción nuevamente
from Bio.SeqIO import *
domain_base3 = []
for rec in parse('base3_dominio.fasta', "fasta"):
    domain_base3.append(str(rec.seq))
```

Se graficó el resultado para tener una idea de los dominios encontrados (Figura 6). Para ello, por la forma en que se construye el gráfico, primero fue necesario extender todas las secuencias con espacios en blanco para que tengan la misma longitud. Se creó una nueva lista con esa característica que se llamó *domain_base3_long*.

```
In [12]:
domain_base3_long = []
domain_base3_long = domain_base3[:]

#Secuencias más larga
seq_longer = 0
for i in domain_base3_long:
    if len(i) > seq_longer:
        seq_longer = len(i)

#Lista con secuencias de igual largo
for n,i in enumerate(domain_base3_long):
    if len(i)<=seq_longer:
            nro_ext = seq_longer-len(i)
    to_ext = ""
    to_ext = " "*nro_ext
    domain_base3_long[n]= (i + to_ext)
```

```
In [13]:
#Graficos
#Comando para que vaya mostrando los gráficos
%matplotlib inline

import sys
from matplotlib import pyplot as pl
import matplotlib.colors as colors
```

```
from matplotlib.lines import Line2D
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
import numpy as np


#Se eligen colores usando el comando colors.cnames
#Se crea un diccionario para asociar dominios a colores
colores= {
'a':'orange',
'b':'violet',
'c':'lightblue',
'd':'tomato',
'e':'mediumaquamarine',
'f':'moccasin',
'g':'peru',
'h':'lightgreen',
'-':'lightgray' }

#Crear un array del largo de la secuencia
equis=np.arange(len(domain_base3_long[0]))
equis

#Graficos
for letra in colores.keys():
    curvas=[]
    for i,prot in enumerate(domain_base3_long):
        curva=[ i+1 if x==letra else -1 for x in prot]
        curvas.append(curva)
    todasLasCurvas=np.array(curvas)
    pl.plot(equis,todasLasCurvas.T,'|',color=colors.cnames
      [colores[letra]])

pl.ylim([0,16])

pl.rcParams["figure.figsize"] = [15,10]

#Titulos y nombres a los ejes
pl.title("Predicciones de dominios", size=18, y = 1.005)
pl.ylabel('Organismo', size = 14)
pl.xlabel('Posicion', size = 14)

#Nombres de organismos
pl.yticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15],name_base3)

#Leyenda
```

```
orange = mpatches.Patch(label="HELICASE_ATP_BIND_1", color="orange")
violet = mpatches.Patch(label="HELICASE_CTER", color="violet")
lightblue = mpatches.Patch(label="DICER_DSRBF", color="lightblue")
tomato = mpatches.Patch(label="PAZ", color="tomato")
mediumaquamarine = mpatches.Patch(label="RNASE_3_2",
color="mediumaquamarine")
moccasin = mpatches.Patch(label="RNASE_3_1", color="moccasin")
peru = mpatches.Patch(label="DS_RBD" , color="peru")
lightgreen = mpatches.Patch(label="None", color="lightgreen")
lightgray = mpatches.Patch(label='Sin dominio', color="lightgray")
legend =
pl.legend(handles=[orange,violet,lightblue,tomato,mediumaquamarine,m
occasin,peru,lightgreen,lightgray], loc=6, bbox_to_anchor=(1, 0.5))

pl.savefig("6_Dominios.png", bbox_inches='tight',
bbox_extra_artist=[legend])
```

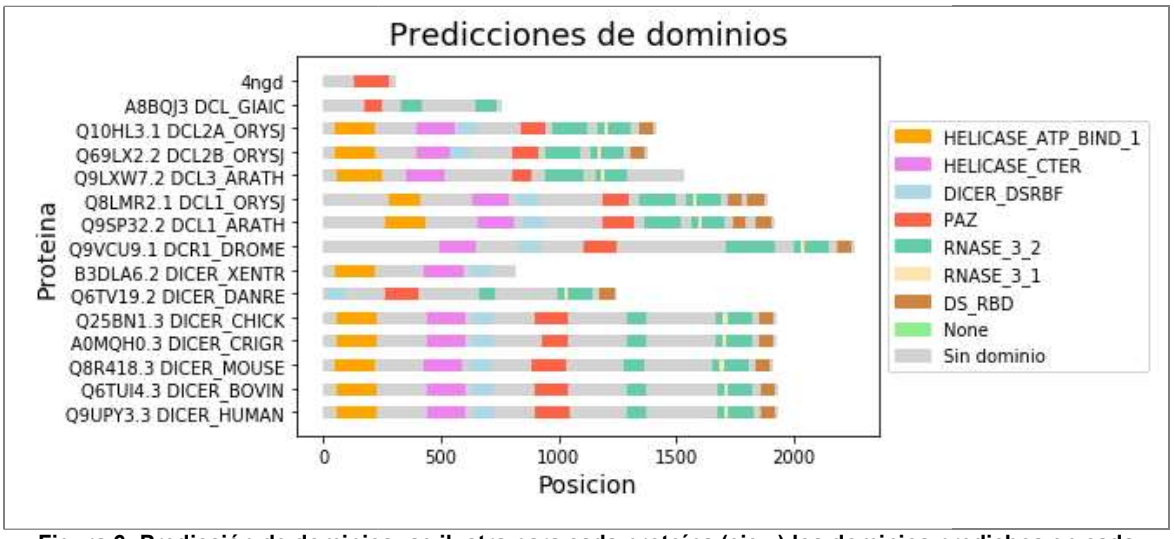


Figura 6: Predicción de dominios: se ilustra para cada proteína (eje y) los dominios predichos en cada posición (eje x). En la leyenda se muestran los nombres y colores para cada dominio predicho.

El dominio "plataforma" no está definido en Prosite, por lo tanto resulta fundamental restringir el área de análisis, y continuar con la predicción y modelado estructural para poder identificarlo en la secuencia de DCL1 de *Arabidopsis thaliana*. Para asegurarse de que la región de interés con la que se continuaría con el análisis contuviese a las regiones Plataforma y PAZ, se decidió extraer de cada secuencia todo lo que se encontraba entre la posición donde termina el dominio DICER_DSRBF y donde empieza RNASE_3_2, es decir, desde la última letra c hasta la primera letra e.

Algunas características que surgen al seleccionar la región de interés son:

- las secuencias B3DLA6.2 DICER_XENTR y A8BQJ3 DCL_GIAIC tienen como último dominio a c, por lo que toman desde c hasta el final
- la secuencia Q9LXW7.2 DCL3_ARATH no tiene dominio c, por lo que toma desde el principio hasta la primer e

```
In [14]:
#Seleccionar solo la región entre c y e
domain_base4 = []
seq_base4 = []
for i,seq in enumerate(domain_base3):
    largo = len(seq)
    pos_c = 0
    pos_e = 0
    for pos,l in enumerate(seq):
        if l == "c":
            pos_c = pos
    for pos,l in enumerate(reversed(seq)):
        if l == "e":
                pos_e = pos
    pos_e = largo - pos_e
    domain_base4.append(seq[pos_c:pos_e])
    seq_base4.append(str(seq_base3[i])[pos_c:pos_e])

#Guardar en archivo secuencias
base4_file = open("base4.fasta", "a")
for i, seq in enumerate(seq_base4):
    base4_file.write(">")
    base4_file.write(name_base3[i])
    base4_file.write("\n")
    base4_file.write(seq_base4[i])
    base4_file.write("\n")
base4_file.close()

#Guardar en archivo dominios
base4_dominios_file = open("base4_dominio.fasta", "a")
for i, seq in enumerate(seq_base4):
    base4_dominios_file.write(">")
    base4_dominios_file.write(name_base3[i])
    base4_dominios_file.write("\n")
    base4_dominios_file.write(domain_base4[i])
    base4_dominios_file.write("\n")
base4_dominios_file.close()
```

```
#Visualización de la primer parte base4.fasta
f = open('base4.fasta', 'r')
file_contents = f.read()
print (file_contents [0:2200])
f.close()

#Visualización de la primer parte base4_dominio.fasta
f = open('base4_dominio.fasta', 'r')
file_contents = f.read()
print (file_contents [0:2200])
f.close()
```

```
>Q9UPY3.3 DICER_HUMAN
YEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRDSYPRPDQPCYLYVIGMVLTTPLPD
ELNFRRRKLYPPEDTTRCFGILTAKPIPQIPHFPVYTRSGEVTISIELKKSGFMLSLQMLEL
ITRLHQYIFSHILRLEKPALEFKPTDADSAYCVLPLNVVNDSSTLDIDFKFMEDIEKSEARI
GIPSTKYTKETPFVFKLEDYQDAVIIPRYRNFDQPHRFYVADVYTDLTPLSKFPSPEYETFA
EYYKTKYNLDLTNLNQPLLDVDHTSSRLNLLTPRHLNQKGKALPLSSAEKRKAKWESLQNKQ
ILVPELCAIHPIPASLWRKAVCLPSILYRLHCLLTAEELRAQTASDAGVGVRSLPADFRYPN
LDFGWKKSIDSKSFISISNSSSAENDNYCKHSTIVPENAAHQGANRTSSLENHDQMSVNCRT
LLSESPGKLHVEVSADLTAINGLSYNQNLANGSYDLANRDFCQGNQLNYYKQEIPVQPTTSY
SIQNLYSYENQPQPSDECTLLSNKYLDGNANKSTSDGSPVMAVMPGTTDTIQVLKGRMDSEQ
SPSI
>Q6TUI4.3 DICER_BOVIN
YEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRESYPRPGQPCYLYVIGMVLTTPLPD
ELNFRRRKLYPPEDTTRCFGILTAKPIPQIPHFPVYTRSGEVTISIELKKSGFTLSLQMLEL
ITRLHQYIFSHILRLEKPALEFKPTDADSAYCVLPLNVVNDSSTLDIDFKFMEDIEKSEARI
GIPSTKYSKETPFVFKLEDYQDAVIIPRYRNFDQPHRFYVADVYTDLTPLSKFPSPEYETFA
EYYKTKYNLDLTNLNQPLLDVDHTSSRLNLLTPRHLNQKGKALPLSSAEKRKAKWESLQNKQ
ILVPELCAIHPIPASLWRKAVCLPSILYRLHCLLTAEELRAQTASDAGVGVRSLPVDFRYPN
LDFGWKKSIDSKSFISIANSSSAENENYCKHSTIVVPENAAHQGANRTSPLENHDQMSVNCR
TLFSESPGKLQIEVSTDLTAINGLSYNKSLANGSYDLANRDFCQGNHLNYYKQEIPVQPTTS
YPIQNLYNYENQPKPSDECTLLSNKYLDGNANTSTSDGSPVTAAVPGTTETGEAPPDRTASE
QSPSPG
>Q8R418.3 DICER_MOUSE
YEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRESYPKPDQPCYLYVIGMVLTTPLPD
```

```
ELNFRRRKLYPPEDTTRCFGILTAKPIPQIPHFPVYTRSGEVTISIELKKSGFTLSQQMLEL
ITRLHQYIFSHILRLEKPALEFKPTGAESAYCVLPLNVVNDSGTLDIDFKFMEDIEKSEARI
GIPSTKYSKETPFVFKLEDYQDAVIIPRYRNFDQPHRFYVADVYTDLTPLSKFPSPEYETFA
EYYKTKYNLDLTNLNQPLLDVDHTSSRLNLLTPRHLNQKGKALPLSSAEKRKAKWESLQNKQ
ILVPELCAIHPIPASLWRKAVCLPSILYRLHCLLTAEELRAQTASDAGVGVRSLPVDFRYPN
LDFGWKKSIDSKSFISTCNSSLAESDNYCKHSTTVVPEHAAHQGATRPSLENHDQMSVNCKR
LPAESPAKLQSEVSTDLTAINGLSYNKNLANGSYDLVNRDFCQGNQLNYFKQEIPVQPTTSY
PIQNLYNYENQPKPSNECPLLSNTYLDGNANTSTSDGSPAVSTMPAMMNAVKALKDRMDSEQ
SPSV
>A0MQH0.3 DICER_CRIGR
YEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRESYPKPDQPCYLYVIGMVLTTPLPD
ELNFRRRKLYPPEDTTRCFGILTAKPIPQIPHFPVYTRSGEVTISIELKKSGFTLSQQMLEL
ITRLHQYIFSHILRLEKPALEFKPTGAESAYCVLPLNVVNDSSTLDIDFTFMEDIEKSEARI
GIPSTKYSKETPFVFKLEDYQDAVIIPRYRNFDQPHRFYVADVYTDLTPLSKFPSPEYETFA
EYYKTKYNLDLTNLNQPLLDVDHTSSRLNLLTPRHLNQKGKALPLSSAEKRKAKWESLQNKQ
ILVPELCAIHPIPASLWRKAVCLPSILYRLHCLLTAEELRAQTASDAGVGVRSLPADFRYPN
LDFGWKKSIDSKSFISTCNSSLAENDNYCKHSTIIVPENAAHQGATRTP
>Q9UPY3.3 DICER_HUMAN
c------------------------------------------------------------
------------------------------------------------------------
---------------------------------------------
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddd----------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
--------------------e
>Q6TUI4.3 DICER_BOVIN
c------------------------------------------------------------
------------------------------------------------------------
---------------------------------------------
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
```

```
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddd--------------------------------
--------------------------------------------------------------
--------------------------------------------------------------
--------------------------------------------------------------
----------------------e
>Q8R418.3 DICER_MOUSE
c-------------------------------------------------------------
--------------------------------------------------------------
---------------------------------------------
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddd--------------------------------
--------------------------------------------------------------
--------------------------------------------------------------
--------------------------------------------------------------
--------------------e
>A0MQH0.3 DICER_CRIGR
c-------------------------------------------------------------
--------------------------------------------------------------
--------------------------------------------------------------
-----------------
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd------------
--------------------------------------------------------------
--------------------------------
```

3.3. Alineado de secuencias

Una vez construida la base de datos de secuencias de proteínas tipo DICER en la región de interés, se procedió a realizar un alineamiento. Dado que la presencia de secuencias divergentes podía complicar el proceso, se decidió comenzar calculando el porcentaje de identidad de secuencia presente entre cada par de las mismas, con lo que se construyó el gráfico que se muestra más abajo (Figura 7), en donde se marcó en rojo las combinaciones que dieron identidades menores al 30%.

```
In [15]:
from Bio import pairwise2
from Bio import SeqIO
import numpy as np
np.set_printoptions(precision=2)

identidades = []
matriz = np.ones((len(seq_base4)+1, len(seq_base4)+1))
for i,seq1 in enumerate(seq_base4):
    for j,seq2 in enumerate(seq_base4[i+1:]):
        alignments = pairwise2.align.globalxx(seq1, seq2)
        identidades.append((alignments[0])[2])
        matriz[i+1,j+i+1+1]=(alignments[0])[2]/
       max((len(seq1),len(seq2)))
        matriz[j+i+1+1,i+1]=(alignments[0])[2]/
       max((len(seq1),len(seq2)))

import csv
with open('base4_identidades.csv', 'w') as csvfile:
    writer = csv.writer(csvfile)
    [writer.writerow(r) for r in matriz]
```

	DICER HUMAN	DICER BOVIN	DICER MOUSE	DICER CRIGR	DICER CHICK	DICER DANRE	DICER XENTR	DCR1 DROME	DCL1 ARATH	DCL1 ORYSJ	DCL3 ARATH	DCL2B ORYSJ	DCL2A ORYSJ	DCL GIAIC	4ngd
DICER HUMAN	1.00	0.94	0.92	0.93	0.90	0.75	0.17	0.38	0.33	0.33	0.29	0.27	0.27	0.27	0.53
DICER BOVIN	0.94	1.00	0.93	0.93	0.90	0.75	0.17	0.38	0.33	0.33	0.29	0.27	0.26	0.27	0.53
DICER MOUSE	0.92	0.93	1.00	0.96	0.88	0.74	0.17	0.38	0.33	0.33	0.29	0.27	0.27	0.27	0.52
DICER CRIGR	0.93	0.93	0.96	1.00	0.90	0.75	0.17	0.38	0.33	0.34	0.29	0.27	0.27	0.27	0.52
DICER CHICK	0.90	0.90	0.88	0.90	1.00	0.76	0.17	0.38	0.33	0.33	0.29	0.27	0.27	0.27	0.52
DICER DANRE	0.75	0.75	0.74	0.75	0.76	1.00	0.15	0.37	0.33	0.31	0.28	0.26	0.26	0.28	0.50
DICER XENTR	0.17	0.17	0.17	0.17	0.17	0.15	1.00	0.10	0.16	0.15	0.09	0.19	0.18	0.17	0.19
DCR1 DROME	0.38	0.38	0.38	0.38	0.38	0.37	0.10	1.00	0.27	0.27	0.33	0.22	0.22	0.22	0.22
DCL1 ARATH	0.33	0.33	0.33	0.33	0.33	0.33	0.16	0.27	1.00	0.74	0.24	0.33	0.33	0.32	0.33
DCL1 ORYSJ	0.33	0.33	0.33	0.34	0.33	0.31	0.15	0.27	0.74	1.00	0.24	0.32	0.32	0.31	0.32
DCL3 ARATH	0.29	0.29	0.29	0.29	0.29	0.28	0.09	0.33	0.24	0.24	1.00	0.20	0.21	0.20	0.20
DCL2B ORYSJ	0.27	0.27	0.27	0.27	0.27	0.26	0.19	0.22	0.33	0.32	0.20	1.00	0.99	0.34	0.38
DCL2A ORYSJ	0.27	0.26	0.27	0.27	0.27	0.26	0.18	0.22	0.33	0.32	0.21	0.99	1.00	0.34	0.37
DCL GIAIC	0.27	0.27	0.27	0.27	0.27	0.28	0.17	0.22	0.32	0.31	0.20	0.34	0.34	1.00	0.34
4ngd	0.53	0.53	0.52	0.52	0.52	0.50	0.19	0.22	0.33	0.32	0.20	0.38	0.37	0.34	1.00

Figura 7: Matriz de identidades

Se decidió eliminar de la base de datos las secuencias que dieron un porcentaje de identidad menor al 25% con DCL1: DICER_XENTR, DCR1_DROME y DCL3_ARATH, siendo la primera de ellas mucho más corta que el resto, y las otras dos más largas. Quedó así conformado lo que se denominó la **base5**. Se construyeron los archivos *base5.fasta* y *base5_dominio.fasta* con las secuencias que se conservaron.

```
In [16]:
#Archivos y las variables sin las secuencias eliminadas para el
alineamiento
print name_base3[6]
print name_base3[7]
print name_base3[10]
```

```
B3DLA6.2 DICER_XENTR
Q9VCU9.1 DCR1_DROME
Q9LXW7.2 DCL3_ARATH
```

```
In [17]:
name_base5=[]
name_base5.extend(name_base3[0:6] + name_base3[8:10] +
name_base3[11:])
seq_base5=[]
seq_base5.extend(seq_base4[0:6] + seq_base4[8:10] + seq_base4[11:])
domain_base5=[]
domain_base5.extend(domain_base4[0:6] + domain_base4[8:10] +
domain_base4[11:])

#Guardar en archivo
base5_file = open("base5.fasta", "a")
for i, seq in enumerate(seq_base5):
    base5_file.write(">")
    base5_file.write(name_base5[i])
    base5_file.write("\n")
    base5_file.write(seq_base5[i])
    base5_file.write("\n")
base5_file.close()


#Guardar en archivo
base5_dominios_file = open("base5_dominio.fasta", "a")
for i, seq in enumerate(seq_base5):
    base5_dominios_file.write(">")
    base5_dominios_file.write(name_base5[i])
    base5_dominios_file.write("\n")
    base5_dominios_file.write(domain_base5[i])
    base5_dominios_file.write("\n")
base5_dominios_file.close()
```

Para el alineamiento se utilizó el programa MEGA versión 5.03 y dentro de este a ClustalW. El alineamiento se exportó como *base5_alineada.fas* Los parámetros utilizados se muestran a continuación en la Figura 8:

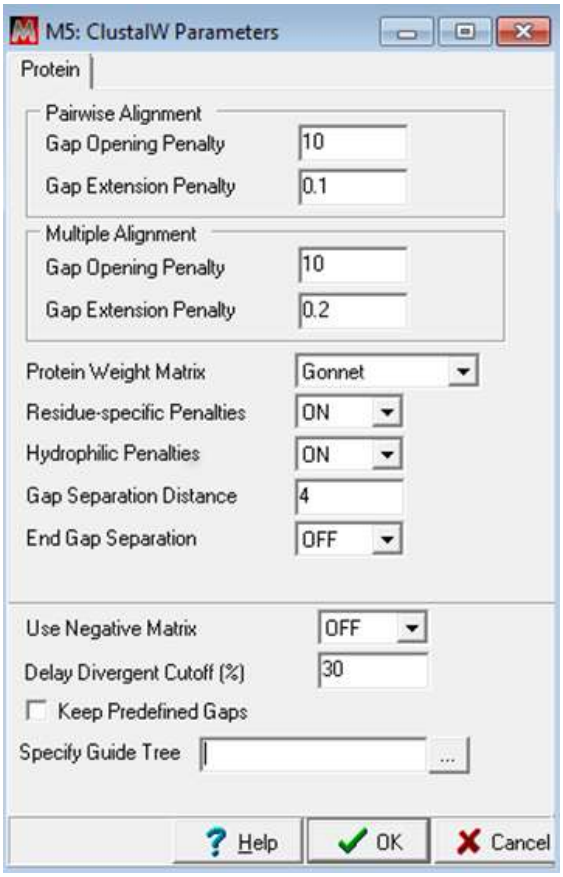


Figura 8: Parámetros MEGA

Continuando con el uso del mismo programa se realizó un análisis de las secuencias alineadas mediante la construcción de un árbol con el Test con Maximun Likelihood con los parámetros estándar. El resultado se muestra en el siguiente árbol (Figura 9):

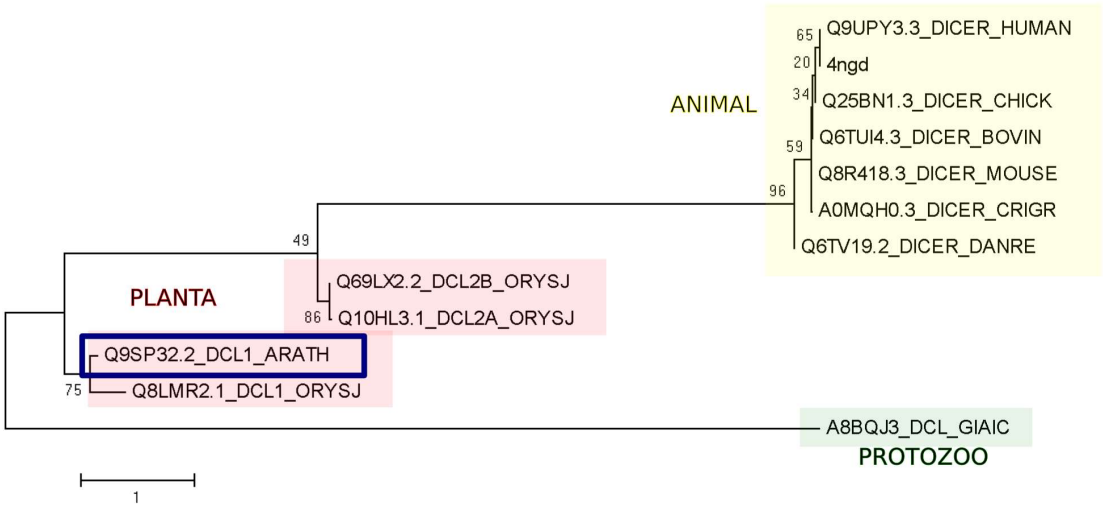


Figura 9: Arbol filogenético

En el árbol obtenido se pudo observar la presencia de miembros de tres reinos:

- animal: donde se destaca la presencia de las variantes de DICER humana en cercanía con el resto de las secuencias animales.
- planta: se encuentran particularmente DCL1_ARATH que es la de interés, y otras que pertenecen a la especie de arroz.
- protozoo: esta variante fue agregada externamente y aparece en un clado aislado.

```
In [18]:
#Visualización de la primer parte de base5_alineada.fasta
f = open('base5_alineada.fas', 'r')
file_contents = f.read()
print (file_contents [0:2200])
f.close()
```

```
>Q9UPY3.3_DICER_HUMAN
YEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRDSYPRPDQPCYLYVIGMVLTTPLPD
ELNFRRRKLYPPEDTTRCFGILTAKPIPQIP--
HFPVYTRSGEVTISIELKKSGFMLSLQMLELITRLHQYIFSHILRLEKPALEFKPTDADSAY
CVLPLNVVNDSS-
TLDIDFKFMEDIEKSEARIGIPSTKYTKETPFVFKLEDYQDAVIIPRYRNFDQPHRFYVADV
```

YTDLTPLSKFPSPEYETFAEYYKTKYNLDLTNLNQPLLDVDHTSSRLNLLTPRHLNQKGKAL
PLSSAEKRKAKWESLQNKQILVPELCAIHPIPASLWRKAVCLPSILYRLHCLLTAEELRAQT
ASDAGVGVRSLPADFRYPNLDFGWKKSIDSKSFISISNSSSAENDNYCKHSTIVP-
ENAAHQGANRTSSLENHDQMSVNCRTLLSESPGKLHVEVSADLTAINGLSYNQNLANGSYDL
ANR-
DFCQGNQLNYYKQEIPVQPTTSYSIQNLYSYENQPQPSDECTLLSNKYLDGNANKSTSDGSP
VMAVMPGTTDTIQVLKGRMDSEQS----------PSI--
>Q6TUI4.3_DICER_BOVIN
YEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRESYPRPGQPCYLYVIGMVLTTPLPD
ELNFRRRKLYPPEDTTRCFGILTAKPIPQIP--
HFPVYTRSGEVTISIELKKSGFTLSLQMLELITRLHQYIFSHILRLEKPALEFKPTDADSAY
CVLPLNVVNDSS-
TLDIDFKFMEDIEKSEARIGIPSTKYSKETPFVFKLEDYQDAVIIPRYRNFDQPHRFYVADV
YTDLTPLSKFPSPEYETFAEYYKTKYNLDLTNLNQPLLDVDHTSSRLNLLTPRHLNQKGKAL
PLSSAEKRKAKWESLQNKQILVPELCAIHPIPASLWRKAVCLPSILYRLHCLLTAEELRAQT
ASDAGVGVRSLPVDFRYPNLDFGWKKSIDSKSFISIANSSSAENENYCKHSTIVVPENAAHQ
GANRTSPLENHDQMSVNCRTLFSESPGKLQIEVSTDLTAINGLSYNKSLANGSYDLANR-
DFCQGNHLNYYKQEIPVQPTTSYPIQNLYNYENQPKPSDECTLLSNKYLDGNANTSTSDGSP
VTAAVPGTTETGEAPPDRTASEQS----------PSPG-
>Q8R418.3_DICER_MOUSE
YEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRESYPKPDQPCYLYVIGMVLTTPLPD
ELNFRRRKLYPPEDTTRCFGILTAKPIPQIP--
HFPVYTRSGEVTISIELKKSGFTLSQQMLELITRLHQYIFSHILRLEKPALEFKPTGAESAY
CVLPLNVVNDSG-
TLDIDFKFMEDIEKSEARIGIPSTKYSKETPFVFKLEDYQDAVIIPRYRNFDQPHRFYVADV
YTDLTPLSKFPSPEYETFAEYYKTKYNLDLTNLNQPLLDVDHTSSRLNLLTPRHLNQKGKAL
PLSSAEKRKAKWESLQNKQILVPELCAIHPIPASLWRKAVCLPSILYRLHCLLTAEELRAQT
ASDAGVGVRSLPVDFRYPNLDFGWKKSIDSKSFISTCNSSLAESDNYCKHSTTVVPEHAAHQ
GATRPS-LENHDQMSVNCKRLPAESPAKLQSEVSTDLTAINGLSYNKNLANGSYDLVNR-
DFCQGNQLNYFKQEIPVQPTTSYPIQNLYNYENQPKPSNECPLLSNTYLDGNANTSTSDGSP
AVSTMPAMMNAVKALKDRMDSEQS----------PSV--
>A0MQH0.3_DICER_CRIGR
YEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRESYPKPDQPCYLYVIGMVLTTPLPD

```
ELNFRRRKLYPPEDTTRCFGILTAKPIPQIP--
HFPVYTRSGEVTISIELKKSGFTLSQQMLELITRLHQYIFSHILRLEKPALEFKPTGAESAY
CVLPLNVVNDSS-
TLDIDFTFMEDIEKSEARIGIPSTKYSKETPFVFKLEDYQDAVIIPRYRNFDQPHRFYVADV
YTDLTPLSKFPSPEYETFAEYYKTKYNLDLTNLNQPLLDVDHTSSRLNLLTPRHLNQKGKAL
PLSSAEKRKAKWESLQNKQILVPELCAIHPIPASLWRKAVCLPSILYRLHCLLTAEELRAQT
ASDAGVGVRSLPA
```

Se parseó el alineamiento, y se usó el resultado para alinear los dominios. Para ello se decidió identificar con el guión medio (-) a los casos donde no hubo predicción de dominios, y como guión bajo (_) a los lugares en donde se incorporó algún gap durante el alineamiento. Finalmente se graficó el resultado obtenido (Figura 10).

```
In [19]:
seq_align_base5 = []
for rec in parse('base5_alineada.fas', "fasta"):
    seq_align_base5.append(str(rec.seq))

#Alineo las secuencias con los dominios
domain_align_base5 = []
i = 0
while i < len(seq_align_base5):
    domain_letters = list(domain_base5[i])
    alignment_letters = list(seq_align_base5[i])
    g = 0
    h = 0
    domain_alig = ""
    while g < len (alignment_letters):
        if alignment_letters[g] == "-":
            domain_alig = domain_alig + "_"
        else:
            domain_alig = domain_alig + domain_letters[h]
            h = h + 1
        g = g + 1
    domain_align_base5.append(domain_alig)
    i = i + 1
```

```
In [20]:
#Guardar en archivo
base5_dominios_alineada_file = open("base5_dominio_alineada.fasta",
"a")
for i, seq in enumerate(seq_base5):
    base5_dominios_alineada_file.write(">")
    base5_dominios_alineada_file.write(name_base5[i])
    base5_dominios_alineada_file.write("\n")
    base5_dominios_alineada_file.write(domain_align_base5[i])
    base5_dominios_alineada_file.write("\n")
base5_dominios_alineada_file.close()

#Visualización de la primer parte de base5_dominio_alineada.fasta
f = open('base5_dominio_alineada.fasta', 'r')
file_contents = f.read()
print (file_contents [0:2200])
f.close()
```

```
>Q9UPY3.3 DICER_HUMAN
c-------------------------------------------------------------
------------------------------__------------------------------
----------------------------------------------_--
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddd----------------------------------
-----------------------------------------------------_--------
-------------------------------------------------------------_----
-------------------------------------------------------------
--------------------____________--e__
>Q6TUI4.3 DICER_BOVIN
c-------------------------------------------------------------
------------------------------__------------------------------
----------------------------------------------_--
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddd----------------------------------
-------------------------------------------------------------
-------------------------------------------------------------_----
-------------------------------------------------------------
--------------------____________---e_
>Q8R418.3 DICER_MOUSE
c-------------------------------------------------------------
------------------------------__------------------------------
----------------------------------------------_--
```

```
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddd--------------------------------
--------------------------------------------------------------
----_-------------------------------------------------------_----
--------------------------------------------------------------
--------------------____________--e__
>A0MQH0.3 DICER_CRIGR
c-------------------------------------------------------------
------------------------------__-----------------------------
-----------------------------------------------_--------------
--------------------
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd------
---------------------------------------
```

```
In [21]:
#Graficos
#Comando para que vaya mostrando los gráficos
%matplotlib inline

import sys
from matplotlib import pyplot as pl
import matplotlib.colors as colors
from matplotlib.lines import Line2D
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
import numpy as np


#Se eligen colores usando el comando colors.cnames
#Se crea un diccionario para asociar dominios a colores
colores= {
'd':'tomato',
'-':'lightgray' }

#Crear un array del largo de la secuencia
equis=np.arange(len(domain_align_base5[0]))
equis

#Graficos
for letra in colores.keys():
    curvas=[]
    for i,prot in enumerate(domain_align_base5):
        curva=[ i+1 if x==letra else -1 for x in prot]
```

```
        curvas.append(curva)
    todasLasCurvas=np.array(curvas)
    pl.plot(equis,todasLasCurvas.T,'|',color=colors.cnames
    [colores[letra]])

pl.ylim([0,13])

pl.rcParams["figure.figsize"] = [15,10]

#Titulos y nombres a los ejes
pl.title("Predicciones de dominios base5", size=18, y = 1.005)
pl.ylabel('Organismo', size = 14)
pl.xlabel('Posicion', size = 14)

#Nombres de organismos
pl.yticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],name_base5)

#Leyenda
tomato = mpatches.Patch(label="PAZ", color="tomato")
lightgray = mpatches.Patch(label='Sin dominio', color="lightgray")
legend = pl.legend(handles=[tomato,lightgray], loc=6,
bbox_to_anchor=(1, 0.5))

pl.savefig("10_Dominios_base5.png", bbox_inches='tight',
      bbox_extra_artist=[legend])
```

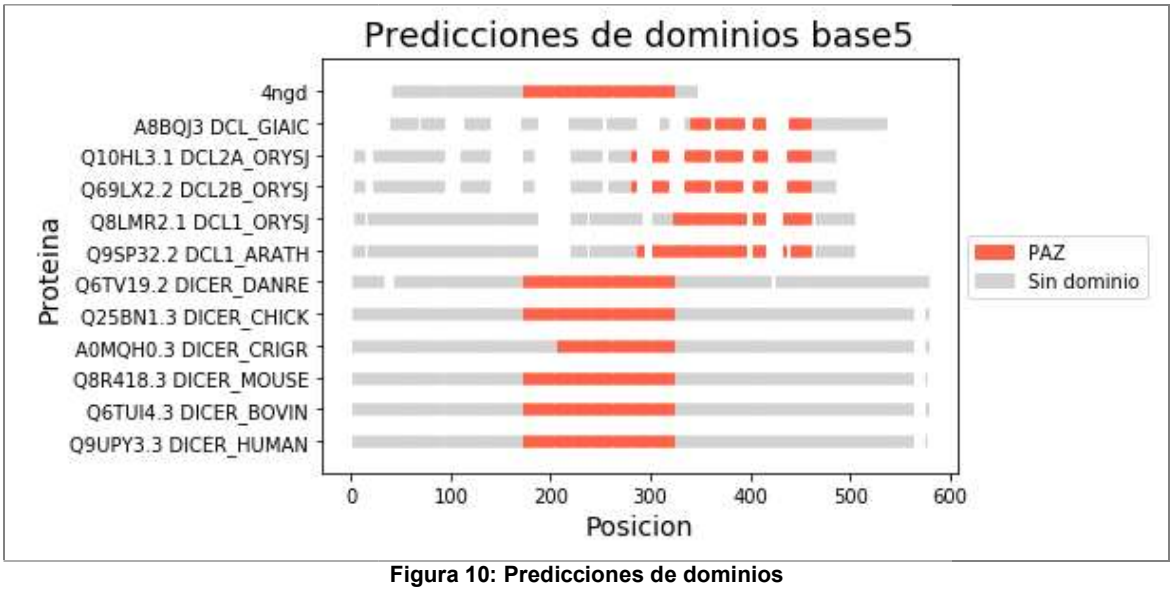


Figura 10: Predicciones de dominios

3.4. Predicción de estructura secundaria

Para la predicción de estructura secundaria se utilizó el programa Jpred (A Protein Secondary Structure Prediction Server), una interfaz disponible en el sitio *http://www.compbio.dundee.ac.uk/jpred4/index_up.html* (Figura 11).

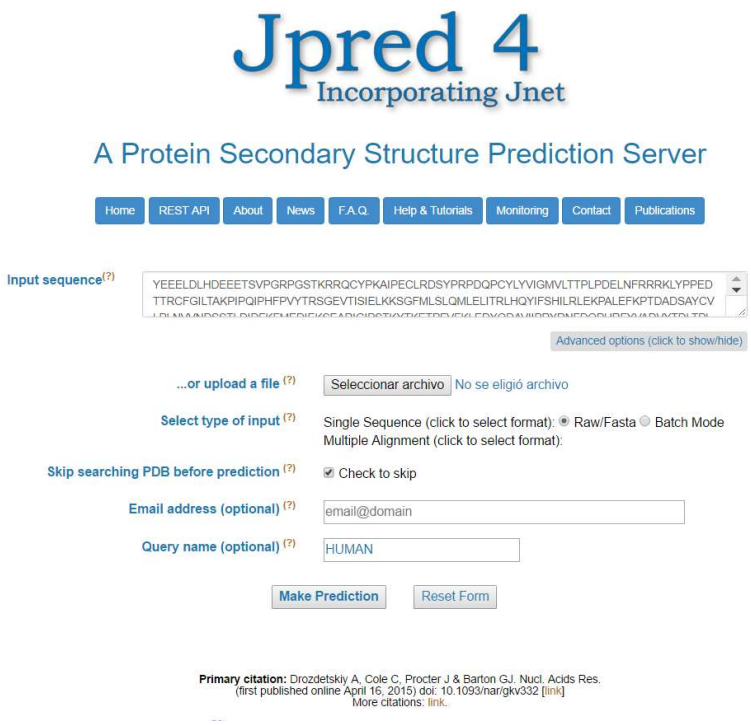


Figura 11: Programa JPred

Por cada secuencia que se carga dentro del servidor, este construye automáticamente un alineamiento múltiple a través de 3 iteraciones del algoritmo de PSI-BLAST. Cuando el programa termina de correr se puede observar el ID de la búsqueda. Se tiene la posibilidad de obtener el resultado en múltiples formatos, y eso se traduce en distintas formas para la URL. Se eligió extraerlo como "Simple JNet Output". Como no se pudo automatizar para todas las secuencias juntas, solo se hizo la predicción para las que contaban con información cristalográfica, las cuales se muestran a continuación junto con sus IDs y largos de secuencia.

```
     Q9UPY3.3 DICER_HUMAN : ID: jp_LVwmmOD
YEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRDSYPRPDQPCYLYVIGMVLTTPLPDELNFRR
RKLYPPEDTTRCFGILTAKPIPQIPHFPVYTRSGEVTISIELKKSGFMLSLQMLELITRLHQYIFSHI
LRLEKPALEFKPTDADSAYCVLPLNVVNDSSTLDIDFKFMEDIEKSEARIGIPSTKYTKETPFVFKLE
DYQDAVIIPRYRNFDQPHRFYVADVYTDLTPLSKFPSPEYETFAEYYKTKYNLDLTNLNQPLLDVDHT
SSRLNLLTPRHLNQKGKALPLSSAEKRKAKWESLQNKQILVPELCAIHPIPASLWRKAVCLPSILYRL
HCLLTAEELRAQTASDAGVGVRSLPADFRYPNLDFGWKKSIDSKSFISISNSSSAENDNYCKHSTIVP
ENAAHQGANRTSSLENHDQMSVNCRTLLSESPGKLHVEVSADLTAINGLSYNQNLANGSYDLANRDFC
QGNQLNYYKQEIPVQPTTSYSIQNLYSYENQPQPSDECTLLSNKYLDGNANKSTSDGSP
VMAVMPGTTDTIQVLKGRMDSEQSPSI largo = 562
     Q8R418.3 DICER_MOUSE jp_a4PrKkv
YEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRESYPKPDQPCYLYVIGMVLTTPLPDELNFRR
RKLYPPEDTTRCFGILTAKPIPQIPHFPVYTRSGEVTISIELKKSGFTLSQQMLELITRLHQYIFSHI
LRLEKPALEFKPTGAESAYCVLPLNVVNDSGTLDIDFKFMEDIEKSEARIGIPSTKYSKETPFVFKLE
DYQDAVIIPRYRNFDQPHRFYVADVYTDLTPLSKFPSPEYETFAEYYKTKYNLDLTNLNQPLLDVDHT
SSRLNLLTPRHLNQKGKALPLSSAEKRKAKWESLQNKQILVPELCAIHPIPASLWRKAVCLPSILYRL
HCLLTAEELRAQTASDAGVGVRSLPVDFRYPNLDFGWKKSIDSKSFISTCNSSLAESDNYCKHSTTVV
PEHAAHQGATRPSLENHDQMSVNCKRLPAESPAKLQSEVSTDLTAINGLSYNKNLANGSYDLVNRDFC
QGNQLNYFKQEIPVQPTTSYPIQNLYNYENQPKPSNECPLLSNTYLDGNANTSTSDGSPAVSTMPAMM
NAVKALKDRMDSEQSPSV largo = 562
     Q9SP32.2 DCL1_ARATH jp_STd3SWJ
AEKADQDDEGEPVPGTARHREFYPEGVADVLKGEWVSSGKEVCESSKLFHLYMHNVRCVDFGSSKDPF
LSEVSEFAILFGNELDAEVLSMSMDLYVARAMITKASLAFKGSLDITENQLSSLKKFHVRLMSIVLDV
DVEPSTTPWDPAKAYLFVPVTDNTSMEPIKGINWELVEKITKTTAWDNPLQRARPDVYLGTNERTLGG
DRREYGFGKLRHNIVFGQKSHPTYGIRGAVASFDVVRASGLLPVRDAFEKEVEEDLSKGKLMMADGCM
VAEDLIGKIVTAAHSGKRFYVDSICYDMSAETSFPRKEGYLGPLEYNTYADYYKQKYGVDLNCKQQPL
IKGRGVSYCKNLLSPRFEQSGESETVLDKTYYVFLPPELCVVHPLSGSLIRGAQRLPSIMRRVESMLL
AVQLKNLISYPIPTSKILE largo = 427
     DCL_GIAIC jp_VO7Gdk8
MHALGHCCTVVTTRGPSHWLLLLDTHLGTLPGFKVSAGRGLPAAEVYFEAGPRVSLSRTDATIVAVYQ
SILFQLLGPTFPASWTEIGATMPHNEYTFPRFISNPPQFATLAFLPLLSPTSPLDLRALMVTAQLMCD
AKRLSDEYTDYSTLSASLHGRMVATPEISWSLYVVLGIDSTQTSLSYFTRANESITYMRYYATAHNIH
LRAADLPLVAAVRLDDLKDHQIPAPGSWDDLAPKLRFLPPELCLLLPDEFDLIRVQALQFLPEIAKHI
CDIQNTICALDKSFPDCGRIGGERYFAITAGLRLDQGRGRGLAGWRTPF GPFGVSH largo =
328
```

Junto con la obtención del resultado fue necesario un paso para eliminar espacios vacíos que se agregaban automáticamente. Tanto las secuencias utilizadas como los resultados de la predicción se guardaron en los archivos *base6.fasta* y *base6_ss.fasta*.

El resultado está disponible en el servidor con los IDs mencionados solo por unos días, por ello lo conveniente es cargar las variables directamente desde los archivos.

```
In [22]:
from urllib2 import urlopen
baseurl = "http://www.compbio.dundee.ac.uk/jpred4/results/"
from pyquery import PyQuery as pq

base6_file=open("base6.fasta", "a")
base6_ss_file=open("base6_ss.fasta", "a")


#DICER_HUMAN (largo 562)
jpredCode = "jp_8vFUwRb"
finalUrl = baseurl + jpredCode + "/" + jpredCode + ".simple.html"
response = urlopen(finalUrl)
response_text_Human = response.read()
d = pq(response_text_Human)
p = d("html body pre code")
seq_human=p.text()[:562]
ss_human=p.text()[563:]

ss_human_sinesp = ""
for i in ss_human:
    if i==" ":
        pass
    else:
        ss_human_sinesp = ss_human_sinesp + i
base6_file.write(">DICER_HUMAN\n")
base6_ss_file.write(">DICER_HUMAN\n")
base6_file.write(seq_human)
base6_ss_file.write(ss_human_sinesp)

#DICER_MOUSE (largo 562)
baseurl = "http://www.compbio.dundee.ac.uk/jpred4/results/"
jpredCode = "jp_JCMWLr_"
finalUrl = baseurl + jpredCode + "/" + jpredCode + ".simple.html"
response = urlopen(finalUrl)
response_text_Mouse = response.read()
d = pq(response_text_Mouse)
p = d("html body pre code")
seq_mouse=p.text()[:562]
ss_mouse=p.text()[563:]

ss_mouse_sinesp = ""
for i in ss_mouse:
    if i==" ":
        pass
    else:
        ss_mouse_sinesp = ss_mouse_sinesp + i
```

```
base6_file.write("\n>DICER_MOUSE\n")
base6_ss_file.write("\n>DICER_MOUSE\n")
base6_file.write(seq_mouse)
base6_ss_file.write(ss_mouse_sinesp)

#DCL1_ARATH (largo 427)
baseurl = "http://www.compbio.dundee.ac.uk/jpred4/results/"
jpredCode = "jp_KNew2z3"
finalUrl = baseurl + jpredCode + "/" + jpredCode + ".simple.html"
response = urlopen(finalUrl)
response_text_DCL1 = response.read()
d = pq(response_text_DCL1)
p = d("html body pre code")
seq_dcl1=p.text()[:427]
ss_dcl1=p.text()[428:]

ss_dcl1_sinesp = ""
for i in ss_dcl1:
    if i==" ":
        pass
    else:
        ss_dcl1_sinesp = ss_dcl1_sinesp + i

base6_file.write("\n>DCL1_ARATH\n")
base6_ss_file.write("\n>DCL1_ARATH\n")
base6_file.write(seq_dcl1)
base6_ss_file.write(ss_dcl1_sinesp)

#DCL_GIAIC (largo 328)
baseurl = "http://www.compbio.dundee.ac.uk/jpred4/results/"
jpredCode = "jp_Hp7khT7"
finalUrl = baseurl + jpredCode + "/" + jpredCode + ".simple.html"
response = urlopen(finalUrl)
response_text_Giaic = response.read()
d = pq(response_text_Giaic)
p = d("html body pre code")
seq_giaic=p.text()[:328]
ss_giaic=p.text()[329:]

ss_giaic_sinesp = ""
for i in ss_giaic:
    if i==" ":
        pass
    else:
        ss_giaic_sinesp = ss_giaic_sinesp + i
```

```
base6_file.write("\n>DCL_GIAIC\n")
base6_ss_file.write("\n>DCL_GIAIC\n")
base6_file.write(seq_giaic)
base6_ss_file.write(ss_giaic_sinesp)

base6_file.close()
base6_ss_file.close()
```

```
In [23]:
from Bio.SeqIO import *

# Parseando los nombres de las secuencias del fasta
name_base6 = []
for rec in parse('base6.fasta', "fasta"):
    name_base6.append(str(rec.description))

# Parseando las secuencias del fasta
seq_base6 = []
for rec in parse('base6.fasta', "fasta"):
    seq_base6.append(str(rec.seq))

# Parseando las ss de las secuencias del fasta
ss_base6 = []
for rec in parse('base6_ss.fasta', "fasta"):
    ss_base6.append(str(rec.seq))

# Parseando las secuencias alineadas
seq_align_base5 = []
for rec in parse('base5_alineada.fas', "fasta"):
    seq_align_base5.append(str(rec.seq))

#Visualización de la primer parte base3.fasta
f = open('base6_ss.fasta', 'r')
file_contents = f.read()
print (file_contents [0:2200])
f.close()
```

```
>DICER_HUMAN
------------------------------E--HHH---------EEEEEEEEEEE------
-----EEE-----------EEEE----------EEEE----EEEEEEE--------
HHHHHHHHHHHHHHHHHHHHH----------------EEEE--------EEE--
HHHHHHHHHH-------------------------EEEEE---------EEEEEE-------
-----------HHHHHHHH--EEEEE----EEEEEE------------------------
-----------EEEEE----------HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH--
--------------------HHHHH-------------------------------------
------------------------------------------------------------
-----------EE----------------------EEE---------------------
-----------------
>DICER_MOUSE
------------------------E-----E--HHH---------EEEEEEEEEE-------
-----EEE-----------EEEE----------EEEE----EEEEEEE--------
HHHHHHHHHHHHHHHHHHHHH----------------------------EEE---
HEEEEH----------------------------EEEEE---------EEEEEE--------
---------HHHHHHHHH--EEEEE----EEEEEE------------------------
----------EEEEE----------HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH----
-------------------HHHHHHH-------------------------------------
-------------------------------------------------E------------
------------------------------------------------------------
----------------
>DCL1_ARATH
---------------------------EEEE----------------EEEEEEEE-------
------------EEEEEE----------EEEEEE----EEEEEEEE--EEE--
HHHHHHHHHHHHHHHHHHH----------------EEEEE------E--------EEEEEE-
-------------HHHHHHHHHH-----------------------------------------
--------------HHHHHHHHH-HH-----------EEEEEEEE-----EEEEEEE-----
-----------------HHHHHHHHHH---EEEE----EEEE--------------------
-------EEEEE-----------HHHHHHHHHHHHHHHHHHHHHHHHHHHHHH---------
--
>DCL_GIAIC
------EEEEEE-----EEEHHHHHH-------EE---------EEEEEE--EEEE-----
```

```
HHHHHHHHHHHHH--------------E---------------------EEE----------
---HHHHHHH----------------HHEEEEEEEEE-----EEEEEEE-----------
-----EEEEEHHH----EEE------EEEEE----------------------EEE-----
EE--------EEE--HHHHHHHHHHHHHHHHHHHHH-------------EEEEEEEEEE--
--------------------
```

Como se puede ver en la Figura 12, al comparar la estructura secundaria de los archivos formato pdb (secuencias superiores) de estructuras cristalográficas (*2ffl.pdb* para DCL_GIAIC y *4ngd.pdb* para DICER_HUMANA) con la estructura secundaria predicha por el programa Jpred (secuencias inferiores), la predicción fue en general buena.

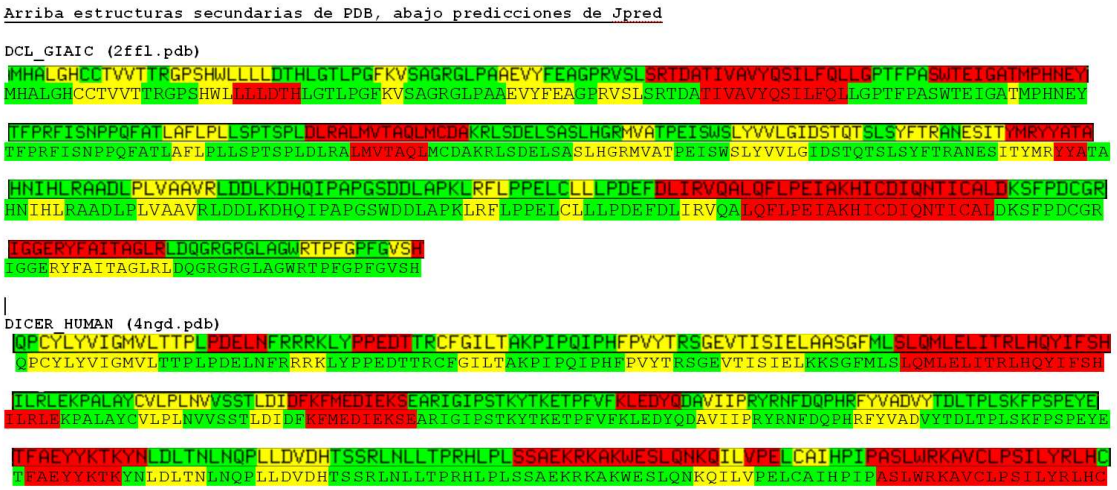


Figura 12: Análisis de estructura secundaria: en amarillo "extended" (E), en rojo "helical" (H) y en verde sin estructura secundaria predicha

Se creó una nueva lista de secuencias alineadas formada solo por las proteínas para las cuales se hizo predicción de estructura secundaria. Luego se incorporaron los alineamientos a la predicción de estructura secundaria, se eliminaron los espacios en los cuales había gaps para todas las secuencias de la lista y finalmente se volvió a graficar (Figura 13).

```
In [24]:
#Base 6 alineamientos

seq_align_base6 = []
seq_align_base6 =
seq_align_base5[0:1]+seq_align_base5[2:3]+seq_align_base5[6:7]+seq_a
lign_base5[10:11]

#Agregando los espacios de los alineamientos
ss_align_base6 = []
i = 0
while i < len(seq_align_base6):
    ss_letters = list(ss_base6[i])
    alignment_letters = list(seq_align_base6[i])
    g = 0
    h = 0
    ss_alig = ""
    while g < len (alignment_letters):
        if alignment_letters[g] == "-":
            ss_alig = ss_alig + "_"
        else:
            ss_alig = ss_alig + ss_letters[h]
            h = h + 1
        g = g + 1
    ss_align_base6.append(ss_alig)
    i = i + 1

#Eliminando gaps comunes
seq_human = list(ss_align_base6[0])
seq_mouse = list(ss_align_base6[1])
seq_arath = list(ss_align_base6[2])
seq_giaic = list(ss_align_base6[3])
seq_human_singap = []
seq_mouse_singap = []
seq_arath_singap = []
seq_giaic_singap = []

for n in range(0,len(seq_human)):
    if seq_human[n]==seq_mouse[n]==seq_arath[n]==seq_giaic[n]=="_":
        pass
    else:
        seq_human_singap.append(seq_human[n])
        seq_mouse_singap.append(seq_mouse[n])
        seq_arath_singap.append(seq_arath[n])
        seq_giaic_singap.append(seq_giaic[n])
```

```
ss_align_singap_base6=[''.join(seq_human_singap),''.join(seq_mouse_s
ingap),''.join(seq_arath_singap),''.join(seq_giaic_singap)]
```

```
In [25]:
#Graficos

#Comando para que vaya mostrando los gráficos
%matplotlib inline

import sys
from matplotlib import pyplot as pl
import matplotlib.colors as colors
from matplotlib.lines import Line2D
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
import numpy as np


#Se eligen 3 colores usando el comando colors.cnames (c->
DICER_DSRBF   d-> PAZ   e-> RNASE_3_2)

#Se cre un diccionario para asociar dominios a colores
colores= {
'H':'blue',
'E':'red',
'-':'gray' }

#Crear un array del largo de la secuencia
equis=np.arange(len(ss_align_singap_base6[0]))
equis

#Graficos
for letra in colores.keys():
    curvas=[]
    for i,prot in enumerate(ss_align_singap_base6):
        curva=[ i+1 if x==letra else -1 for x in prot]
        curvas.append(curva)
    todasLasCurvas=np.array(curvas)
    pl.plot(equis,todasLasCurvas.T,'|',color=colors.cnames
           [colores[letra]])

pl.ylim([0,5])

pl.rcParams["figure.figsize"] = [10,5]

#Titulos y nombres a los ejes
```

```
pl.title("Predicciones de estructura", size=18, y = 1.005)
pl.xlabel('Posicion', size = 14)
pl.ylabel('Organismo', size = 14)

#Se elimina los recuadros
ax1=pl.axes()
ax1.set_frame_on(False)

#Escalas visibles solo en x e y
ax1.get_xaxis().tick_bottom()
ax1.get_yaxis().tick_left()
xmin, xmax = ax1.get_xaxis().get_view_interval()
ymin, ymax = ax1.get_yaxis().get_view_interval()


#Lineas horizontales
ax1.add_artist(Line2D((xmin, xmax), (ymin, ymin), color='black',
linewidth=2))
ax1.add_artist(Line2D((xmin, xmin), (ymin, ymax), color='black',
linewidth=2))

#Nombres de organismos
pl.yticks([1, 2, 3, 4, 5],['DICER_HUMAN',
'DICER_MOUSE','DCL1_ARATH', 'DCL_GIAIC'])

#Leyenda
blue = mpatches.Patch(label='H', color="blue")
red = mpatches.Patch(label='E', color="red")
gray = mpatches.Patch(label='Sin SS', color="gray")

legend = pl.legend(handles=[blue,red,gray], loc=6,
bbox_to_anchor=(1, 0.5))

pl.savefig("13_SS.png", bbox_inches='tight',
bbox_extra_artist=[legend])
```

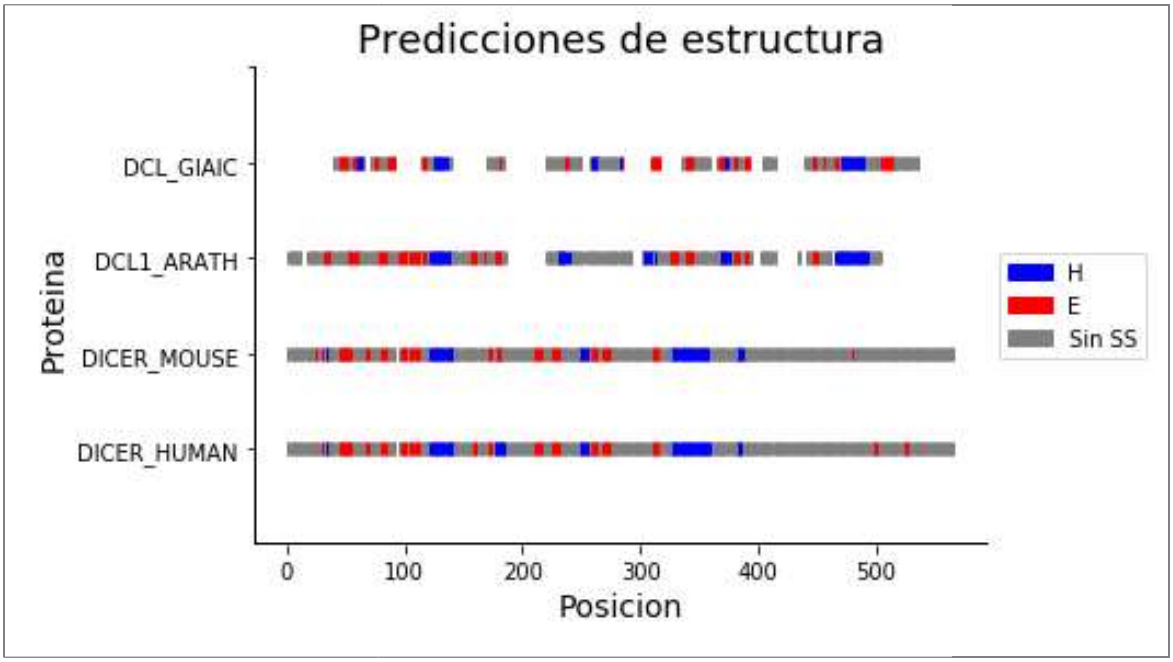


Figura 13: Predicciones de estructura secundaria: en azul "helical" (H), en rojo "extended" (E) y en gris sin estructura secundaria predicha (Sin SS)

A partir del gráfico se observa que las estructuras secundarias, al igual que los dominios, no alinean bien.

3.5. Gráfico conjunto

Finalmente se procedió a graficar las predicciones de estructura secundaria y de dominios juntos (Figura 14).

```
In [26]:
ss_dominios=[]
ss_dominios=[domain_align_base5[0], ss_align_base6[0],
     domain_align_base5[1], domain_align_base5[2],
     ss_align_base6[1], domain_align_base5[3],
     domain_align_base5[4], domain_align_base5[5],
     domain_align_base5[6], ss_align_base6[2],
     domain_align_base5[7], domain_align_base5[8],
     domain_align_base5[9], domain_align_base5[10],
     ss_align_base6[3], domain_align_base5[11]]
```

```
In [27]:
#Gráficos
#Comando para que vaya mostrando los gráficos
%matplotlib inline

import sys
from matplotlib import pyplot as pl
import matplotlib.colors as colors
from matplotlib.lines import Line2D
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
import numpy as np


#Se eligen 3 colores usando el comando colors.cnames (c->
DICER_DSRBF   d-> PAZ   e-> RNASE_3_2)

#Se crea un diccionario para asociar dominios a colores
colores= {
'H':'blue',
'E':'red',
'd':'violet',
'_':'white',
'-':'gray' }


#Crear un array del largo de la secuencia
equis=np.arange(len(ss_dominios[0]))
equis

#Graficos
for letra in colores.keys():
    curvas=[]
    for i,prot in enumerate(ss_dominios):
        curva=[ i+1 if x==letra else -1 for x in prot]
        curvas.append(curva)
    todasLasCurvas=np.array(curvas)
    pl.plot(equis,todasLasCurvas.T,'|',color=colors.cnames
    [colores[letra]])

pl.ylim([0,17])
pl.xlim([0,583])

#Titulos y nombres a los ejes
pl.title("Predicciones de estructura", size=18, y = 1.005)
```

```
pl.xlabel('Posicion', size = 14)
pl.ylabel('Organismo', size = 14)

#Lineas horizontales
ax1.add_artist(Line2D((xmin, xmax), (ymin, ymin), color='black',
linewidth=2))
ax1.add_artist(Line2D((xmin, xmin), (ymin, ymax), color='black',
linewidth=2))

#Nombres de organismos
pl.yticks([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16],['DICER_HUMAN',
'DICER_HUMAN', 'DICER_BOVIN', 'DICER_MOUSE', 'DICER_MOUSE',
'DICER_CRIGR', 'DICER_CHICK', 'DICER_DANRE',
'DCL1_ARATH','DCL1_ARATH', 'DCL1_ORYSJ', 'DCL2B_ORYSJ',
'DCL2A_ORYSJ', 'DCL_GIAIC','DCL_GIAIC', '4ngd'])

#Leyenda
blue = mpatches.Patch(label='H', color="blue")
red = mpatches.Patch(label='E', color="red")
gray = mpatches.Patch(label='Sin prediccion', color="gray")
violet = mpatches.Patch(label='PAZ', color="violet")

legend = pl.legend(handles=[blue,red,gray,violet], loc=6,
bbox_to_anchor=(1, 0.5))

pl.rcParams["figure.figsize"] = [20,20]
pl.savefig("14_SS_Dominios.png", bbox_inches='tight',
bbox_extra_artist=[legend])
```

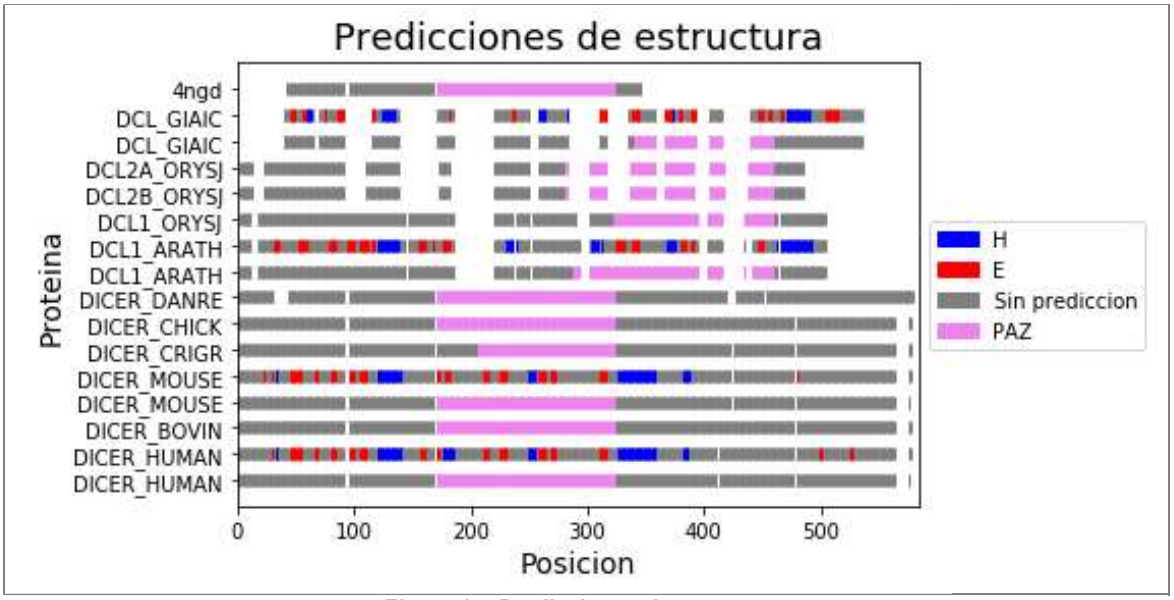


Figura 14: Predicciones de estructura

3.6. Alineamiento estructural

Para el alineamiento estructural se extrajeron las secuencias alineadas de las proteínas DCL1 de *Arabidopsis thaliana* y DICER humana, trabajando con el archivo fasta del alineamiento y utilizando el módulo de AlignIO. Luego se las guardó en un archivo .phy eliminando previamente las posiciones en las cuales ambas secuencias presentaban gaps (producto de que eran resultado de un alineamiento múltiple).

```
In [28]:
from Bio import AlignIO

seq_alineadas=AlignIO.read("base5_alineada.fas","fasta")
my_seq=seq_alineadas[6:7]
my_seq.append(seq_alineadas[0])

print my_seq
```

```
SingleLetterAlphabet() alignment with 2 rows and 580 columns
-AEKADQDDEGEP----VPGTARHREFYPEGVADVLKGEWVSSG...---
Q9SP32.2_DCL1_ARATH
YEEELDLHDEEETSVPGRPGSTKRRQCYPKAIPECLRDSYPRPD...I--
Q9UPY3.3_DICER_HUMAN
```

```
In [29]:
#Registrar sin los dobles gaps
my_seq_correct=my_seq[:,0:2]
for columna in range(2,my_seq.get_alignment_length()):
    if not my_seq.get_column(columna)=="--":
        my_seq_correct=my_seq_correct+my_seq[:,columna:columna+1]

#Guardar en un archivo
AlignIO.write(my_seq_correct,"DICER_DCL1.phy","phylip")
```

```
d:\programas\python\lib\site-
packages\Bio\Align\__init__.py:622:
BiopythonDeprecationWarning: This method is deprecated and is
provided for backwards compatibility with the old
Bio.Align.Generic.Alignment object. Please use the slice
notation instead, as get_column is likely to be removed in a
future release of Biopython.
  "of Biopython.", BiopythonDeprecationWarning)
     Out[36]:
     1
```

Se utilizó el programa Modeller, que está diseñado para el modelado por homología estructural (tutorial: *https://salilab.org/modeller/tutorial/basic.html*). Tomando como base el archivo .phy creando anteriormente, se adaptó para poder ser leído por el programa convirtiéndolo en un archivo de tipo .ali con el nombre *4ngd_DCL1.ali*. Para ello se conservaron solo las posiciones que estaban presentes en el archivo 4ngd.pdb junto con su alineamiento en DCL1 (zona con marca gris), y para los lugares en los que el PDB no tenía estructura pero que debían estar ocupados por residuos (regiones en negrita) se incorporaron manualmente gaps en forma de "-", finalmente se consideraron también dos mutaciones puntuales que tenía la secuencia del molde con respecto a su secuencia *wildtype* (KK subrayada) (Figura 15).

```
 2 567
Q9SP32.2_D -AEKADQDDE GEP----VPG TARHREFYPE GVADVLKGEW VSSGKEVCES
Q9UPY3.3_D YEEELDLHDE EETSVPGRPG STKRRQCYPK AIPECLRDSY PRPDQPCYLY

           SKLFHLYMHN VRCVDFGSSK DPFLSEVSEF AILFGNELDA EVLSMSMDLY
           VIGMVLTTPL PDELNFRRRK LYPPEDTTRC FGILTAKPIP QIP--HFPVY

           VARAMITKAS LAFKGSLDIT ENQLSSLKKF HVRLMSIVLD VDVEP-STTP
           TRSGEVTISI ELKKSGFMLS LQMLELITRL HQYIFSHILR LEKPALEFKP

           WDPAKAYLFV PVTDNTSMEP IKGINWELVE KITKTTA--- ----------
           TDADSAYCVL PLNVVNDSS- TLDIDFKFME DIEKSEARIG IPSTKYTKET

           ---------- ---------- WDNPLQRARP DVYLGTN--E RTLGGDRREY
           PFVFKLEDYQ DAVIIPRYRN FDQPHRFYVA DVYTDLTPLS KFPSPEYETF

           G-FGKLRHNI VFGQKSHPTY GIRGAVASFD VVRASGLLPV RDAF------
           AEYYKTKYNL DLTNLNQPLL DVDHTSSRLN LLTPRHLNQK GKALPLSSAE

           -EKEVEEDLS KGKLMMADGC MVAEDLIGKI VTAAHSGKRF YVDSICYDMS
           KRKAKWESLQ NKQILVPELC AIHPIPASLW RKAVCLPSIL YRLHCLLTAE
```

Figura 15: Alineamiento para la construcción del archivo .ali

```
In [30]:
from modeller import *
from modeller.automodel import *
env = environ()
a = automodel(env, alnfile='DICER_DCL1.ali',
              knowns='4ngd', sequence='DCL1_DICER',
              assess_methods=(assess.DOPE,
                              assess.GA341))
a.starting_model = 1
a.ending_model = 1
a.make()
```

```
      check_ali___> Checking the sequence-structure alignment.
[…]
>> Summary of successfully produced models:
Filename                          molpdf     DOPE score    GA341 score
------------------------------------------------------------------------
DCL1_DICER.B99990001.pdb      1338.95593   -20412.89062        0.06761
```

El resultado obtenido fue un archivo llamado *DCL1_DICER.B99990001*. Se muestran en la salida otros tres parámetros: *molpdf* que es una medida de la suma de restricciones, *DOPE score* (Discrete Optimized Protein Energy) que es una estadística en función de energía y *GA341* que tiene en cuenta el porcentaje de identidad entre el templado y el modelo. Los tres sirven para seleccionar la mejor estructura entre una colección de modelos generados por MODELLER, por lo que en

este caso no se analiza porque solo se generó un modelo. En la Figura 16 se muestra el mismo comparado con el original *4ngd.pdb* visualizados en el programa Pymol (4ngd en rosa, DCL1 en celeste, se marcaron más oscuro las regiones que corresponden al dominio PAZ de 4ngd, y su correspondiente alineada en DCL1).

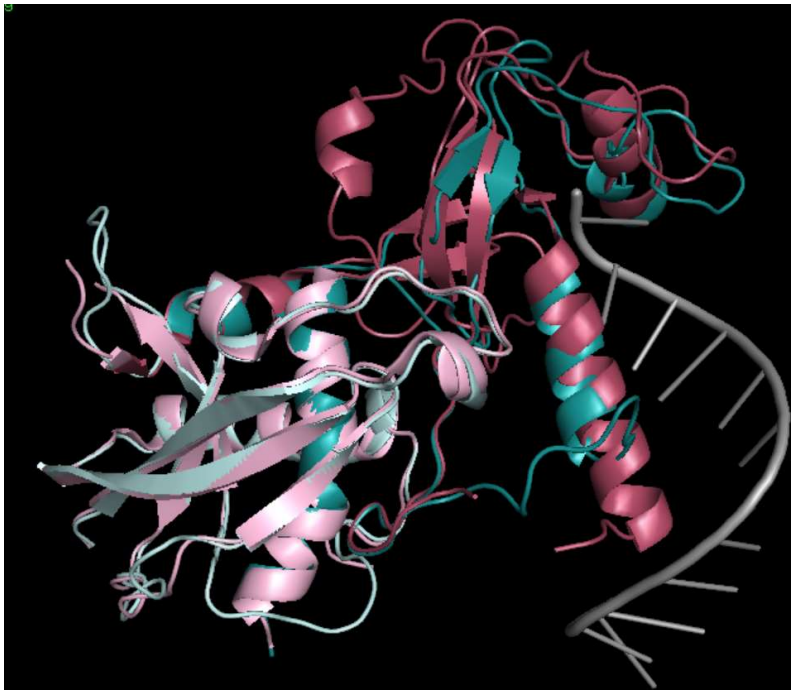


Figura 16a: Estructuras 3D (4ngd en rosa, DCL1 en celeste)

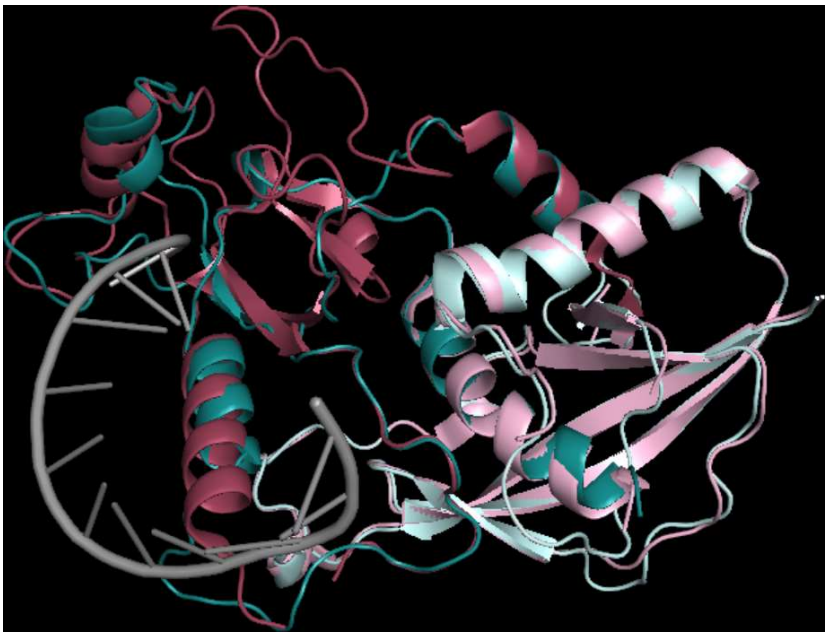


Figure 16b: Estructuras 3D (4ngd en rosa, DCL1 en celeste)

4. CONCLUSIONES

Como conclusiones generales de este trabajo:

- se pudieron construir varias bases de datos con distintas características, todas ellas formadas por proteína de tipo DICER, y siempre conteniendo las variables de interés DCL1 de *Arabidopsis thaliana* y DICER humana.
- se llevó a cabo la predicción de dominios para las secuencias que se pudieron representar fácilmente en un gráfico. A partir de los mismos se seleccionó una región de interés para cada una de las secuencias.
- se seleccionaron secuencias con las que realizó un alineamiento.
- se llevó a cabo la predicción de estructura secundaria la cual fue analizada nuevamente mediante la construcción de un gráfico.
- se hizo un alineamiento estructural entre una región de la proteína DICER humana que había sido cristalografiada previamente con un fragmento de ARN y una región homóloga de la proteína DCL1 de *Arabidopsis thaliana*.

Si bien se pudo cumplir con los objetivos propuestos, los alineamientos no fueron buenos al ser analizados en función de las predicciones de dominios y de estructuras secundarias. La causa principal para esto es que, a diferencia de lo que se esperaba en un principio, no se halló un continuo en homología de secuencia a través de los distintos organismos de origen que pudiera guiar el alinemiento global entre ambas secuencias de proteínas. El resultado se podría optimizar mediante la generación de nuevas bases de datos, y trabajando principalmente en una mejora del alineamiento con el uso de otros parámetros y programas. Sin embargo, considerando la variedad de procedimientos y resultados distintos obtenidos durante el desarrollo de este trabajo, se puede decir que el resultado final presentado satisface el objetivo inicial.

5. REFERENCIAS BIBLIOGRÁFICAS

Bologna, N. G., Mateos, J. L., Bresso, E. G., y Palatnik, J. F. (2009) A loop-to-base processing mechanism underlies the biogenesis of plant microRNAs miR319 and miR159, *EMBO J* **28**, 3646–56.

Burdisso, P., Suarez, I. P., Bologna, N. G., Palatnik, J. F., Bersch, B., Rasia, R. M. (2012) Second Double-Stranded RNA Binding Domain of Dicer-like Ribonuclease 1: Structural and Biochemical Characterization, *Biochemistry* **51**, 10159-10166.

Du, Z., Lee, J. K., Tjhen, R., Stroud, R. M., y James, T. L. (2008) Structural and biochemical insights into the dicing mechanism of mouse Dicer: a conserved lysine is critical for dsRNA cleavage, *Proc Natl Acad Sci USA* **105**, 2391–6.

MacRae, I. J., Zhou, K., Li, F., Repic, A., Brooks, A. N., Cande, W. Z., Adams, P. D. y Doudna, J. A. (2006) Structural Basis for Double-Stranded RNA Processing by Dicer, *Science* **311**, 195-198.

Mateos, J. L., Bologna, N. G., Chorostecki, U., y Palatnik, J. F. (2010) Identification of MicroRNA processing determinants by random mutagenesis of Arabidopsis MIR172a precursor, *Curr Biol* **20**, 49–54.

miRBase: the microARN database, *http://www.mirbase.org/*

Rana, T. M. (2007) Illuminating the silence: understanding the structure and function of small RNAs, *Nat Rev Mol Cell Biol* **8**, 23–36.

Song, L., Axtell, M. J., y Fedoroff, N. V. (2010) RNA secondary structural determinants of miRNA precursor processing in Arabidopsis, *Curr Biol* **20**, 37–41.

Tian. Y., Simanshu, D. K., Ma, J.-B., Park, J.-E., Heo, I., Kim, V. N. y Patel, D. J. (2014) A Phosphate-Binding Pocket within the Platform-PAZ-Connector Helix Cassette of Human Dicer, *Mol Cell* **53**, 606–616.

Weinberg, D. E., Nakanishi, K., Patel, D. J., y Bartel, D. P. (2011) The Inside-Out Mechanism of Dicers from Budding Yeasts, *Cell* **146**, 262–276.

Werner, S., Wollmann, H., Schneeberger, K., y Weigel, D. (2010) Structure determinants for accurate processing of miR172a in Arabidopsis thaliana, *Curr Biol* **20**, 42–8.

Xie, Z., Khanna, K., y Ruan, S. (2010) Expression of microRNAs and its regulation in plants, *Seminars in Cell & Developmental Biology* 21, 790–797.