Trabajo especial de la Licenciatura en Ciencias de la Computación

Diseño de vacunas atenuadas con menor probabilidad de sufrir reversión a la virulencia

Autor: Santiago Videla
Directora: Laura Alonso Alemany

Resumen

Las denominadas vacunas vivas o atenuadas han sido ampliamente utilizadas para prevenir enfermedades como la rubeola, la poliomielitis, el sarampión o la fiebre amarilla. Sin embargo, uno de los peligros del uso de este tipo de vacunas es la probabilidad de reversión a la virulencia, produciendo la enfermedad que intentan prevenir.

Este trabajo consiste en el análisis, formalización e implementación en un software de una solución para minimizar la probabilidad de reversión virulencia de una vacuna atenuada. Nuestra propuesta consiste en maximizar el número de mutaciones necesario para que la secuencia de RNA del virus vacunal llegue a convertirse en una secuencia semejante a las patógenas o revertantes, siempre manteniendo las propiedades que le otorgan la atenuación.

Clasificación: Biology and genetics, Health, Combinatorial algorithms.

Palabras claves: Bioinformática, Biología computacional, Diseño racional de vacunas atenuadas, Optimización combinatoria basada en restricciones, Búsqueda local.

Universidad Nacional de Córdoba

Trabajo especial de la Licenciatura en Ciencias de la Computación

Diseño de vacunas atenuadas con menor probabilidad de sufrir reversión a la virulencia

Autor:
Santiago Videla

Directora:
Laura Alonso Alemany

21 de diciembre de 2010

Resumen

Las denominadas vacunas vivas o atenuadas han sido ampliamente utilizadas para prevenir enfermedades como la rubeola, la poliomielitis, el sarampión o la fiebre amarilla. Sin embargo, uno de los peligros del uso de este tipo de vacunas es la probabilidad de reversión a la virulencia, produciendo la enfermedad que intentan prevenir.

En trabajos previos se han desarrollado varios enfoques para reducir la probabilidad de reversión, tales como priorizar el uso de determinados codones o pares de codones, sin modificar la secuencia aminoacídica[6] o seleccionar polimerasas más fidedignas[20].

En este trabajo se propone un enfoque complementario, que consiste en maximizar el número de mutaciones necesario para que la secuencia de Ácido ribonucleico (RNA) del virus vacunal llegue a convertirse en una secuencia semejante a las patógenas o revertantes, siempre manteniendo las propiedades que le otorgan la atenuación.

Hemos analizado y formalizado una solución para este problema, y la hemos implementado en una herramienta de software. Esta herramienta es altamente configurable, con lo cual puede tratar diferentes virus. Como prueba de concepto del software, se ha aplicado a la vacuna oral contra la poliomielitis. En este caso, la atenuación viene dada por la estructura secundaria de su RNA, con lo cual se ha configurado el software específicamente para conservar este aspecto.

Prefacio

Cuando empezamos con este trabajo no estaba muy convencido de su relevancia para las Ciencias de la Computación, probablemente por estar enmarcado dentro de lo que conocemos como "ciencia aplicada" y por cómo muchas veces se catalogan estos trabajos como "más simples" que los de la "ciencia pura".

Durante mi paso por la Universidad, pude participar de diferentes discusiones sobre el rol de la ciencia y la tecnología en función de un proyecto político nacional y popular. Creo que una de las conclusiones a las que pudimos llegar con mis compañeros de militancia es que el debate no se trata de "ciencia pura vs. ciencia aplicada", sino más bien, ¿ciencia para qué y para quién?.

En ese sentido creo que es indispensable un trabajo interdisciplinario que permita generar nuevos conocimientos y aplicarlos para la resolución de problemas concretos de la sociedad. Las unidades académicas aisladas y encerradas en sus propias disciplinas, no pueden más que tender a quedar incomunicadas de lo que las rodean.

Por otro lado, creo que los esfuerzos individuales de hacer ciencia "comprometida" se diluyen y quedan en la nada si no hay un compromiso del Estado en priorizar e impulsar las áreas estratégicas que se consideren más relevantes para el desarrollo del país. Este trabajo es probablemente una prueba de ello.

> *Ese valor potencial que tiene cualquier descubrimiento científico es el que tendría un ladrillo arrojado en cualquier lugar del país, si a alguno se le ocurriera construir allí una casa, por casualidad. Es posible, pero no se puede organizar una sociedad, ni la ciencia de un país con ese tipo de criterio.*
>
> *(Oscar Varsavsky, 1920 - 1976)*

Si el Estado no impulsa la producción pública de vacunas y medicamentos, difícilmente este trabajo signifique un aporte al desarrollo nacional. Por el contrario, es mucho más probable que sea aprovechado por cualquier otro país o empresa farmacéutica, a quienes seguramente poco les importan estas líneas "politizadas".

Ojalá este trabajo, que por el momento y con un poco de suerte, es simplemente un "ladrillo arrojado en cualquier lugar del país", sirva algún día para construir los cimientos de un sistema científico menos preocupado por los *papers* publicados en revistas internacionales y más dedicado al desarrollo de una Ciencia Nacional, Latinoamericana y Popular.

Agradecimientos

A Laura Alonso i Alemany, a Daniel Gutson y a Daniel Rabinovich por dirigirme y colaborar en este trabajo, cada uno desde su lugar y siempre dispuestos a ayudarme en todo lo que hizo falta.

A Javier Blanco, a Franco Luque y a Renato Cherini por formar parte del tribunal a esta altura del año, pero sobre todo por haberme acompañado durante toda la carrera.

A todos mis compañeros de militancia con los que aprendí tantas cosas que no se enseñan en el aula.

A mi familia por bancarme siempre.

Índice general

Parte I

Preliminares

Capítulo 1

Introducción

> A foolish faith in authority is the worst enemy of truth.
>
> Albert Einstein

1.1. Para los ansiosos

El objetivo de este trabajo es el diseño y desarrollo de un software que sirva como soporte para el diseño de vacunas atenuadas. En este sentido, la propuesta es encontrar un conjunto de secuencias de RNA que conserven las propiedades que le otorgan la atenuación a la vacuna y que al mismo tiempo, tiendan a maximizar el número de mutaciones necesarias para alcanzar secuencias semejantes a las patógenas o revertantes[1].

En los capítulos 2 y 3 se introducen los conceptos biológicos básicos y algunos de los problemas característicos de la bioinformática, profundizando en aquellos que están más relacionados con este trabajo. Luego, en el capítulo 4 se describen la metodología clásica para el diseño de vacunas atenuadas, algunos antecedentes del diseño racional y finalmente, la solución propuesta. Por último, en la parte III se describen los detalles puramente técnicos y más relevantes sobre el desarrollo del software.

1.2. Sobre las vacunas

Existen diferentes posiciones sobre la efectividad de las vacunas en la prevención de enfermedades. Por un lado, la *versión oficial* sostiene que representan la principal herramienta para combatir enfermedades y epidemias. Pero al mismo tiempo, hay quienes aseguran que las vacunas son, en muchos casos, las causantes de las enfermedades que intentan prevenir. Se cuestionan además, sus ingredientes tóxicos (aluminio, mercurio, cloroformo), en algunos casos cancerígenos o supuestamente relacionados con diferentes enfermedades como el autismo.

[1] (Para el lector ajeno a la biología) que producen la enfermedad que la vacuna debiera prevenir.

Aun así, su uso está ampliamente aceptado en la mayoría de los países del mundo, siendo las campañas masivas de vacunación, una de las principales políticas públicas de salud.

No es el objetivo de este trabajo, profundizar en este tema ni tampoco llegar a una conclusión apresurada sobre la bondad de las vacunas, sino simplemente hacer mención a que es un debate abierto. El lector interesado, podrá consultar la bibliografía tanto a favor como en contra del uso masivo de vacunas según lo crea conveniente.

Un cacho de historia

El origen de las vacunas se remonta al año 1796 durante la epidemia del virus de la viruela en Europa. El médico rural, Edward Jenner, observó que las mujeres que ordeñaban las vacas eventualmente contraían una especie de "viruela vacuna" por el contacto con las ubres, y que luego la viruela común no les producía ningún efecto. Efectivamente, el virus Vaccinia que se usó como vacuna contra la viruela común, resultó ser un virus muy emparentado pero que no producía efectos de consideración en las personas y que dió origen a lo que hoy se conocen como vacunas vivas o atenuadas.

1.3. Motivación

Dejando de lado la discusión planteada anteriormente, el objetivo básico de una vacuna es estimular el sistema inmune sin producir la enfermedad en cuestión. En este sentido, las vacunas atenuadas presentan algunas ventajas frente a las denominadas vacunas inactivas.

- Proveen inmunidad a largo plazo.
- Bajos costos.
- Pocas dosis son suficientes para adquirir inmunidad.
- Fáciles de aplicar.

Sin embargo, este tipo de vacunas también presenta una importante desventaja, como la probabilidad de revertir a la virulencia, lo que da origen a este trabajo.

En sucesivas replicaciones, un virus atenuado puede acumular mutaciones en su secuencia de RNA que le devuelvan su carácter patogénico, produciendo la enfermedad que se deseaba prevenir. A diferencia de los virus Ácido desoxirribonucleico (DNA), los virus RNA poseen una alta frecuencia de mutaciones estimada en 0.1 a 10 mutaciones por genoma replicado[20].

Un caso paradigmático es el de la vacuna Sabin contra la poliomielitis, también conocida como *Oral Polio Vaccine* (OPV). Esta vacuna, desarrollada por Albert Sabin en 1957, es una vacuna atenuada administrada por vía oral para prevenir la poliomielitis.

A raíz de una campaña impulsada por la *World Health Organization* (WHO) en 1988 y utilizando la OPV, hacia finales de 2002 se había logrado interrumpir la transimisión endémica del poliovirus en 209 de los 216 países del mundo[2]. Sin embargo, la alta inestabilidad genética de esta vacuna dió lugar a un nuevo grupo

de virus conocidos como poliovirus circulantes derivados de la vacuna (cVDPV), que presentan propiedades similares a las del poliovirus salvaje, incluyendo neurovirulencia[2] y responsables de una nueva enfermedad denominada parálisis poliomielítica asociada a la vacuna oral (VAPP).

Diferentes estudios muestran que VAPP ocurre con una tasa de aproximadamente un caso cada 750,000 a 1 millon de niños que reciben la primer dosis de OPV[2]. Más aún, en Estados Unidos entre 1980 y 1999, el 95 % de los casos registrados de poliomielitis paralitica fueron VAPP[13].

En Argentina, el último caso registrado de VAPP se produjo en el año 2009 en la provincia de San Luis[7]. Esto derivó en un Alerta Epidemiológico acompañado de fuertes campañas de vacunación con la vacuna OPV.

1.4. Antecedentes

Ante estos datos, se reconoce que uno de los obstáculos para erradicar la poliomielitis es la OPV en si misma[5]. Luego, surge la necesidad de buscar nuevas formas de diseñar vacunas atenuadas que estén libres de riesgos, o al menos, tengan menor probabilidad de revertir a la virulencia.

Es a partir de esto y de los avances en la virología molecular, que empieza a tomar fuerza la idea de **racionalizar el diseño de vacunas atenuadas**, de forma tal de poder controlar y cuantificar la atenuación de las vacunas[14].

Algunos de los métodos propuestos y que analizaremos con mayor detalle más adelante, son la deoptimización de codones[6] y la fidelidad en la replicación del virus atenuado[20].

1.5. Propuesta

En este trabajo, realizado con la colaboración de la Fundación para el Desarrollo de la Programación en Ácidos Nucleicos (FuDePAN)[3], se propone la implementación de un software, que hemos dado en llamar *Combinatory Vaccine Optimizer* (vac-o), para el diseño de secuencias de RNA que optimicen las vacunas atenuadas como un problema de **"optimización combinatoria basado en restricciones"**.

Este tipo de problemas consiste en asignar valores a un conjunto finito de variables que satisfagan determinadas condiciones o restricciones. Estas variables conforman las "componentes de la solución", y las combinaciones de los distintos valores que puede tomar cada componente forman las potenciales soluciones del problema. Luego, usando una función de evaluación sobre las soluciones, se debe encontrar una solución, o varias, que maximicen o minimicen dicha función.[12].

En nuestro caso, las restricciones serán propiedades sobre partes de la secuencia de RNA, como la conservación de la secuencia aminoacídica y la estructura secundaria de RNA. Las soluciones serán secuencias completas de RNA y la función de evaluación sobre estas soluciones, que deseamos maximizar, estará dada por el número de mutaciones necesario para alcanzar una secuencia revertante.

[2]Tendencia o capacidad de un microorganismo de afectar el sistema nervioso.

[3]`http://www.fudepan.org.ar`

En este sentido, la principal innovación que presenta este trabajo, es la utilización de diferentes algoritmos para la predicción de la estructura secundaria de RNA, con el fin de determinar secuencias que conserven parte de la estructura secundaria del virus atenuado y, en consecuencia, mantengan la atenuación y las propiedades como vacuna.

Para guiar el desarrollo del trabajo se tomó como caso testigo la vacuna OPV. Esto nos permitió establecer una serie de requerimientos básicos, que esperamos sean de utilidad para otras vacunas.

Desde el punto de vista de la implementación, se puso especial atención en utilizar diferentes principios y patrones de *Object Oriented Programming* (OOP) con el fin de lograr un software que sea altamente modular y que permita ser extendido en el futuro con nuevas funcionalidades. El código fuente fue liberado bajo licencia *GNU General Public License v3* (GPLv3) y puede ser accedido a través del repositorio Subversion (SVN)[4]

[4] `http://vac-o.googlecode.com`

Parte II

La Biología y la Ciencia de la Computación

Capítulo 2

Biología

> Essentially, all models are wrong, but some are useful.
>
> George E. P. Box

Siendo este un trabajo desde la Ciencia de la Computación, sería absurdo intentar abordar rigurosamente todos los conceptos biológicos involucrados en, por ejemplo, la replicación de un virus dentro de una célula. Al mismo tiempo, para poder comprender la complejidad del problema que este trabajo se propone resolver y las decisiones que se tomaron a la hora de implementar el software, es necesario contar con ciertas definiciones y conceptos biológicos elementales que veremos a continuación.

2.1. Lo esencial

Más allá de la complejidad que existe dentro de un célula, sus componentes y funcionamiento, se destacan tres macro moléculas fundamentales compuestas por moléculas pequeñas:

- DNA y RNA - compuestas por nucleótidos.
- Proteína - compuesta por aminoácidos.

El dogma central

El dogma central de la biología molecular establece que la información fluye del DNA al RNA y luego a las proteínas.

$$\textbf{DNA} \longrightarrow \textbf{RNA} \longrightarrow \textbf{Proteínas}$$

Por supuesto, éste es un modelo extremadamente simplificado al que le faltan componentes fundamentales para que todo funcione como corresponde. Podemos empezar diciendo que las flechas del modelo, implican transformaciones moleculares y, por lo tanto, existen entidades encargadas de llevar adelante estas transformaciones.

La entidad que transforma el DNA en RNA se denomina RNA polimerasa, y el proceso de transformación se conoce como *transcripción*. Por otro lado, la entidad que transforma el RNA en proteínas se denomina ribosoma, y el proceso de esta transformación es conocido como *traducción*.

Ácidos nucleicos

Ambos DNA y RNA son, como mencionamos anteriormente, macro moléculas compuestas por moléculas pequeñas. Estas moléculas pequeñas que los componen se denominan nucleótidos. Para definir en detalle a los nucleótidos, deberíamos profundizar en conceptos químicos que no son relevantes para este trabajo. Sin embargo, podemos decir que en el DNA aparecen 4 nucleótidos, Adenina (A), Guanina (G), Timina (T) y Citosina (C), mientras que en el RNA la T es reemplazada por Uracilo (U). Estos nombres, se corresponden con la base nitrogenada que conforma cada nucleótido y que es lo que diferencia uno del otro. Luego, tanto el DNA como el RNA suelen describirse por su secuencia de nucleótidos.

El DNA se presenta como una doble cadena de nucleótidos, en la que las dos hebras están unidas a través de las bases complementarias (reglas de Watson-Crick[1]), A con T y C con G.

En contraste con el DNA, el RNA se presenta como una cadena simple de nucleótidos. No obstante esto, el RNA también puede plegarse siguiendo las reglas de Watson-Crick (A-U, C-G) e inclusive usando pares menos estables como A-G. Esto da lugar a lo que se conoce como la estructura secundaria y que veremos con mayor detenimiento más adelante.

El concepto de "estabilidad" que se acaba de mencionar esta relacionado con la cantidad de energía libre que se genera como consecuencia de la unión de dos bases de nucleótidos. Generalmente, predominan las uniones entre bases con menor energía libre (más estables), aunque ésto no siempre es así.

Proteínas

Al igual que lo ácidos nucleicos, las proteínas son macro moléculas compuestas por moléculas pequeñas, pero en este caso las moléculas que las componen, se denominan aminoácidos. También al igual que con los nucleótidos, dejaremos de lado la descripción química de los aminoácidos y nos centraremos en los aspectos que son más relevantes para este trabajo.

Existen 20 aminoácidos diferentes que suelen representarse por su nombre abreviado (usando tres letras o una letra). Las proteínas son esencialmente secuencias de al menos 50 aminoácidos y son responsables de diferentes tareas fundamentales para el correcto funcionamiento de la célula. Sin ir mas lejos, el proceso de traducción, es decir, la síntesis de proteínas, lo realizan los ribosomas, que están compuestos en parte por proteínas[2].

Del DNA a las proteínas

Como ya vimos más arriba, el dogma central nos dice que la información en la célula fluye desde el DNA hasta convertirse en proteínas a través de transfor-

[1] Fracis H. Crick y James D. Watson recibieron en 1962 el Premio Nobel en Fisiología o Medicina.

[2] El lector se preguntará ¿qué fue primero, el ribosoma o la proteína?.

maciones moleculares denominadas *transcripción* y *traducción*. También vimos, muy brevemente, que tanto el DNA como el RNA están compuestos por nucleótidos mientras que las proteínas están compuestas por aminoácidos. No profundizaremos en los procesos de transcripción y traducción, pero sí nos interesa ver como se traducen los nucleótidos a los aminoácidos.

Para esto necesitamos introducir el concepto de código genético. El código genético es, precisamente, el conjunto de normas o reglas con las cuales el material genético (secuencia de nucleótidos) se traduce en proteínas (secuencia de aminoácidos). El código define una relación entre secuencias de 3 nucleótidos, llamadas codones, y los aminoácidos. A cada codon le corresponde a lo sumo un aminoácido, pero como veremos en seguida, cada aminoácido puede estar relacionado con más de un codon.

Como vimos anteriormente, en el RNA aparecen 4 nucleótidos. Luego, la cantidad de codones posibles, está dada por $4^3 = 64$ codones posibles. Por otro lado, dijimos que existen 20 aminoácidos, lo que rápidamente nos hace notar que la relación definida por el código genético, no puede ser inyectiva. En criollo, la misma secuencia de aminoácidos puede ser codificada por distintas secuencias de nucleótidos.

La tabla de código genético indica qué codones se traducen en qué aminoácido. Hay aminoácidos que son representados por tan sólo un codon, mientras otros, son representados hasta por 6 codones. Más adelante, veremos que una de las estrategias para racionalizar el diseño de vacunas atenuadas, se basa en determinar qué codones es conveniente usar para codificar una secuencia de aminoácidos determinada, y cómo esto afecta la atenuación del virus.

Además, debemos mencionar la existencia de cuatro codones especiales. Estos codones, uno denominado de *START* y los otros tres denominados de *STOP*, sirven como indicadores para el proceso de transformación desde el RNA hasta las proteínas (traducción).

Esto último es fundamental para delimitar 2 tipos de regiones sobre una secuencia de RNA:

- Regiones traducidas: se las denomina *Open Reading Frame* (ORF) y son las partes de la secuencia que efectivamente se traducen en proteínas. Puede haber uno o varios ORF, pero cada uno debe empezar con un codon de *START* y terminar con uno de *STOP*.

- Regiones no traducidas: Se las denomina *Untranslated Region* (UTR) y se encuentran a izquierda y derecha de un ORF. Se suele hablar de un 5'-UTR y un 3'-UTR, para hacer referencia al extremo izquierdo y derecho respectivamente[3].

Por último, haremos referencia al *Internal Ribosomal Entry Site* (IRES). Un IRES es una secuencia de nucleótidos que permite el inicio del proceso de traducción y suele encontrarse en el 5'-UTR de los virus RNA. Esta secuencia y la estructura secundaria que formen, determina en gran medida la síntesis de proteínas del virus.

[3]La notación de 5' y 3' viene de la notación utilizada en química para numerar los carbonos de un compuesto orgánico.

2.2. La estructura secundaria de RNA

Como ya mencionamos, el plegamiento de una secuencia de RNA entre sus bases complementarias, determina lo que se denomina *estructura secundaria de RNA*. Conocer la estructura secundaria es fundamental para comprender el funcionamiento de los distintos tipos de RNA y de la célula en general. Existen diferentes tipos de RNA pero se distinguen 3 tipos principales:

- *messenger RNA* (mRNA).
- *ribosomal RNA* (rRNA).
- *transfer RNA* (tRNA).

Cada uno de estos tipos de RNA desarrolla diferentes funciones en el modelo del dogma central con el fin de lograr las transformaciones moleculares de transcripción y traducción. Por lo tanto, los plegamientos o estructuras secundarias que forman determinan en gran medida la actividad celular.

La existencia del término *estructura secundaria* nos hace suponer que existe también una *estructura primaria*. De hecho, ya vimos la estructura primaria de RNA cuando dijimos que el RNA se suele representar como una secuencia de nucleótidos. Efectivamente, a esta secuencia se la denomina estructura primaria. También existe la estructura terciaria de RNA, pero la dejaremos de lado por no ser relevante en este trabajo.

Definición 1. Una estructura primaria de RNA S, es una secuencia de RNA de longitud n, $A = a_1a_2a_3 \dots a_n$ con $a_i \in \{A, U, G, C\}$

Definición 2. Dada una estructura primaria o secuencia de RNA de longitud n, la estructura secundaria es un conjunto S de pares (i, j) con $1 \leq i < j \leq n$ tal que para todo, $(i, j), (i', j') \in S$ se satisfacen las tres siguientes condiciones:

- $j - i > 3$
- $i = i' \Leftrightarrow j = j'$
- $i < i' \Rightarrow i < i' < j' < j \vee i < j < i' < j'$

Nótese que cada base de la secuencia puede estar a lo sumo "apareada" o "unida" con una sola base, o eventualmente con ninguna. Además, esta definición supone la ausencia de *pseudo-knots*. Un *pseudo-knot* está formado por 2 pares de bases superpuestos, es decir, (i, j) y (i', j') con $i < i' < j < j'$. Esta suposición se debe principalmente a que de otra manera, los algoritmos que veremos mas adelante para la predicción de estructura secundaria, pasan de tener complejidad polinomial a ser NP-completos[15]. Sin embargo, esta suposición está justificada porque la ocurrencia de *pseudo-knots* es poco frecuente en la naturaleza y porque además pueden ser considerados en análisis posteriores[22].

En la Figura 2.1 se pueden ver las posibles situaciones que se pueden dar para dos pares $(i, j), (i', j')$ con $i < i'$. Según la definición que dimos más arriba, las situaciones en A y B están permitidas mientras que la situación en C representa un *pseudo-knot* y estos no son tenidos en cuenta.

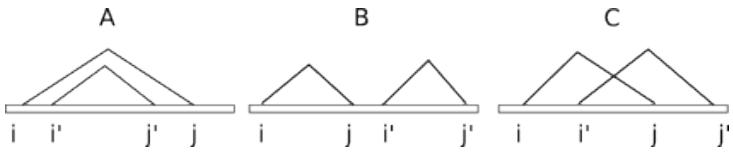

Figura 2.1: A: $i < i' < j' < j$ B: $i < j < i' < j'$ C: $i < i' < j < j'$

La estructura secundaria, el IRES y la atenuación de un virus

La importancia de la estructura secundaria en este trabajo radica, fundamentalmente, en su participación en la síntesis de proteínas del virus y en consecuencia, en su atenuación. Para diferentes virus RNA, se puede decir que la estructura secundaria del IRES determina la "afinidad" o "atracción" con los ribosomas. Luego, esto incide fuertemente en el proceso de traducción de las proteínas que causan la enfermedad, o que están involucradas en el proceso de replicación del virus. Una estructura secundaria con menor afinidad (que la estructura secundaria del virus), le permitiría al sistema inmune generar los anticuerpos necesarios para protegerse del virus antes de que se produzca la enfermedad.

2.3. Los virus RNA

Un virus RNA es, esencialmente, aquel que tiene el RNA como su material genético. Además, los virus suelen clasificarse según la "Clasificación de Baltimore" que los agrupa en diferentes clases (clase I a clase VII) según el tipo de genoma. En particular, en este trabajo se contemplan los virus RNA clase IV también llamados *positive-sense single-stranded RNA virus* ((+)ssRNA virus)

Como ya se mencionó en la sección 1.3, una de las características de este tipo de virus es su alta frecuencia de mutaciones. Esto se debe, principalmente, a la alta tasa de error en su RNA polimerasa[4] (involucrada en el proceso de replicación del virus), estimada entre 1×10^{-3} a 1×10^{-5} errores por nucleótido por ciclo de replicación[20]. Debido a que los virus de RNA suelen tener menos de 10,000 nucleótidos, esto se traduce en 0.1 a 10 mutaciones por genoma replicado.

Si retomamos lo dicho en la sección 2.2 sobre la "afinidad" entre la estructura secundaria y los ribosomas, y cómo esto impacta en la atenuación del virus, se puede ver que esta alta frecuencia de mutaciones, conduce a la posibilidad de que en sucesivas replicaciones, el virus sufra mutaciones que le devuelvan su estructura secundaria original (o similar) y en consecuencia, pierda su atenuación.

Poliovirus

El poliovirus es un (+)ssRNA virus de aproximadamente 7,500 nucleótidos de longitud, miembro del género *Enterovirus* de la familia *Picornaviridae* y causante de la poliomielitis. El poliovirus puede atacar el sistema nervioso y destruir las células nerviosas encargadas del control de los músculos. Como consecuencia, los músculos afectados dejan de cumplir su función y se puede

[4] A diferencia de la DNA polimerasa que posee la capacidad de detectar y corregir errores.

llegar a una parálisis irreversible. En casos severos, la enfermedad puede conducir a la muerte.

La vacuna OPV

Ya presentamos en la sección 1.3 la vacuna OPV contra la poliomielitis y sus principales complicaciones. Con los conceptos vistos hasta el momento, estamos en condiciones de profundizar sobre el por qué de estas complicaciones.

Se han identificado tres serotipos[5] de poliovirus: Poliovirus tipo 1 (PV1), Poliovirus tipo 2 (PV2) y Poliovirus tipo 3 (PV3). Los tres serotipos son extremadamente virulentos y producen los mismos síntomas de la enfermedad. Para cada uno de estos serotipos, se desarrolló su correspondiente virus atenuado Sabin 1, Sabin 2 y Sabin 3 respectivamente, que luego fueron combinados en la vacuna OPV.

Cada uno de estos virus atenuados presenta diferentes niveles de riesgo o probabilidad de revertir a la virulencia. El 90 % de los casos registrados de VAPP son causados por Sabin 2 o Sabin 3, mientras que tan sólo el 10 % restante es causado por Sabin 1[18]. Esto se atribuye, principalmente, a la cantidad de bases de nucleótidos en que difiere cada serotipo respecto a su correspondiente virus atenuado. Mientras que la atenuación en Sabin 2 y Sabin 3 está dada por el impacto de dos o a lo sumo tres mutaciones, la atenuación en Sabin 1 es mas compleja y esto justificaría su menor probabilidad de revertir a la virulencia[18].

Más allá de estas diferencias, los tres virus atenuados, Sabin 1, Sabin 2 y Sabin 3 presentan mutaciones en la 5'-UTR que contribuyen a la atenuación. Además, de los aproximadamente 740 nucleótidos de la 5'-UTR, los primeros 620 se conservan en todos los poliovirus, mientras que las 100 bases que preceden el ORF son las más divergentes. Todo esto sugiere que la 5'-UTR tiene un rol muy importante en el ciclo de vida de los poliovirus[18].

[5] A los fines prácticos y relevantes en este trabajo, formas en que se presenta un determinado virus.

Capítulo 3

Bioinformática y Biología computacional

> I can't be as confident about computer science as I can about biology. Biology easily has 500 years of exciting problems to work on. It's at that level.
>
> Donald Knuth

Por tratarse de una disciplina relativamente nueva, nos permitimos una breve introducción, definición y repaso de los principales problemas abordados por la bioinformática profundizando en aquellos problemas que están directamente relacionados con este trabajo.

3.1. Bio*

La biología siempre dependió en gran parte de la química para poder avanzar y esto dió lugar a lo que se conoce como *bioquímica*. Análogamente, la necesidad de explicar fenómenos biológicos a nivel atómico, dió lugar a la *biofísica*. La *biomatemática* por su parte, se enfoca en el modelado de procesos biológicos utilizando técnicas matemáticas que permitan simular y predecir su comportamiento. La enorme cantidad de datos recopilados por los biólogos y la necesidad de herramientas para interpretarlos, dió origen a lo que hoy conocemos como *bioinformática*.

La bioinformática fue precedida por lo que se llamó *biología computacional*. Aunque no hay una definición precisa para ninguno de los dos términos, la biología computacional se caracterizó por enfocarse en los aspectos teóricos/formales de la Ciencia de la Computación, mientras que la bioinformática supo estar más relacionada con el procesamiento de grandes volúmenes de datos, usualmente almacenados en la Internet. Actualmente, se utilizan ambos términos de manera indiferente.

Problemas clásicos

A continuación presentamos algunos de los problemas o temas de investigación abordados por la bioinformática.

- Análisis de secuencias de DNA o RNA.
- Diseño de secuencias de DNA o RNA.
- Predicción de interacción entre proteínas.
- Análisis de mutaciones en el cáncer.
- Biología evolutiva computacional.

3.2. Predicción de estructura secundaria

Uno de los problemas que omitimos mencionar anteriormente, y que describiremos con mayor detalle, es el que se conoce como "predicción de estructura secundaria". Este problema consiste en, dada una estructura primaria o secuencia de RNA, determinar la estructura secundaria correspondiente. Además, diferentes secuencias de RNA pueden tener la misma estructura secundaria, por lo que también nos interesa determinar para una estructura secundaria dada, las secuencias de RNA que conservan esa estructura.

Por lo mencionado en la secciones 2.2 y 2.3, predecir la estructura secundaria de una secuencia de RNA y conocer las posibles secuencias que conservan una estructura secundaria determinada es de gran importancia en este trabajo.

En inglés, se suele denominar a estos dos problemas *folding* e *inverse folding* respectivamente. A continuación describimos brevemente las diferentes aproximaciones a cada uno de ellos.

3.2.1. Predicción directa (*folding*)

Existen esencialmente 2 tipos de algoritmos para determinar o predecir la estructura secundaria de una secuencia de RNA.

- **Predicción por *minimal free energy* (mfe)**. Propuesto e implementado por Michael Zuker en 1981[23], utiliza programación dinámica para encontrar la estructura secundaria que minimiza la energía libre.
- **Predicción comparativa**. Utiliza diferentes métodos para comparar secuencias y estructuras con el fin de obtener una estructura por "consenso"[9].

Si bien la "predicción comparativa" presenta un incremento en la fidelidad de los resultados obtenidos con respecto a la "predicción por mfe" [9], este tipo de algoritmos requieren la existencia de un conjunto de secuencias relacionadas entre sí (homólogas) y esto no siempre es posible. En particular para este trabajo nos interesa poder realizar predicciones de la estructura secundaria a partir de una sola secuencia, por lo que la "predicción comparativa" fue descartada.

Entre las implementaciones de la "predicción por mfe", se destacan **RNAfold**[11] y **Mfold**[23]. Ambas implementan el algoritmo propuesto por Michael Zuker con complejidad $\mathcal{O}(N^3)$ donde N es la longitud de la secuencia, aunque **RNAfold**

se presenta como una versión mejorada y mas eficiente en la práctica. Como ya mencionamos en la sección 2.2, para lograr esta complejidad polinomial, es necesario suponer la ausencia de *pseudo-knots*. De lo contrario, está demostrado que el problema es NP-completo[15].

A continuación se puede ver un ejemplo de una predicción realizada con **RNAfold**.

```
sancho@mulata:~$ RNAfold

Input string (upper or lower case); @ to quit
....,....1....,....2....,....3....,....4....,....5....,....6....,.
AAAGGCAACGGCCAU
length = 15
AAAGGCAACGGCCAU
...(((....)))..
 minimum free energy =  -4.40 kcal/mol
```

El resultado obtenido es, precisamente, la energía libre y la estructura secundaria representada con paréntesis y puntos, donde los pares de paréntesis indican las bases "apareadas" o "unidas" y los puntos, las bases libres.

3.2.2. Predicción inversa (*inverse folding*)

Este problema consiste en determinar las secuencias de RNA que tienen una estructura secundaria determinada y es fundamental para el diseño racional de secuencias de RNA.

Las principales implementaciones que abordan este problema, lo plantean como un *Constraint Satisfaction Problem* (CSP) y utilizan variantes de búsqueda local estocástica para resolverlo. La función objetivo y que se desea minimizar, es la distancia estructural[1] entre la estructura mfe de la secuencia solución[2] y la estructura secundaria dada.

Si bien la complejidad de este problema no está determinada, a diferencia de otros CSP, la evaluación de las posibles soluciones es muy costosa ya que implica la "predicción mfe" sobre cada secuencia candidata ($\mathcal{O}(N^3)$). Luego, se deben utilizar diferentes técnicas que tiendan a minimizar el número de predicciones realizadas sobre la secuencia completa.

En general, las diferentes implementaciones tienen como parámetros de entrada la estructura secundaria y opcionalmente, una secuencia de RNA incompleta (algunas bases indefinidas, generalmente representadas con la letra N). Se distinguen dos pasos o etapas principales, que marcan las diferencias entre una y otra implementación:

1. **Inicialización:** determinar una secuencia inicial completando la secuencia dada como parámetro. En el caso de no recibir ninguna secuencia como parámetro, se asume una secuencias con todas las bases indefinidas.

[1] Existen diferentes métodos y algoritmos para calcular la distancia entre estructuras cuya descripción exceden este trabajo.

[2] En este contexto, una secuencia solución es aquella cuya predicción de estructura secundaria da como resultado la estructura buscada.

2. **Búsqueda local:** mejorar iterativamente la secuencia inicial hasta alcanzar una secuencia solución. Es decir, realizar cambios en la secuencia que tiendan a minimizar la distancia estructural con la estructura buscada.

Entre estas implementaciones, se destacan **RNAinverse**[11], **INFO-RNA**[4] y **RNA-SSD**[1]. Las últimas dos, se presentan como mejoras a la primera proponiendo diferentes formas de generar la secuencia inicial e implementando algoritmos de búsqueda local estocástica mas complejos.

Nótese que las tres implementaciones dependen de la generación de números *pseudo aleatorios* y ya que la "semilla" utilizada es variable (salvo **RNA-SSD** que permite fijar la semilla), se debe tener en cuenta el no determinismo. Es decir, en sucesivas ejecuciones del algoritmo y utilizando los mismos parámetros, se podrían obtener distintos resultados. Esto sera clave a la hora de sistematizar y automatizar el uso de estos algoritmos.

A continuación, mostramos un ejemplo de una predicción inversa usando **RNAinverse** sobre la estructura obtenida anteriormente con **RNAfold**.

```
sancho@mulata:~$ RNAinverse

Input structure & start string (lower case letters for const
positions) @ to quit, and 0 for random start string
....,....1....,....2....,....3....,....4....,....5....,....6....,..
...(((....)))..
aaaNNNNNNNNNNu
length = 15
aaaUAGUCAGCUACu    3 0
```

Nótese que la secuencia obtenida conserva las bases que ya estaban definidas en la secuencia incompleta, que se dio como parámetro y que además, es distinta a la secuencia sobre la que se hizo la predicción directa anteriormente.

Capítulo 4

Diseño de vacunas atenuadas

> Science is always wrong. It never solves a problem without creating ten more.
>
> George Bernard Shaw

Una vacuna atenuada es aquella que es creada reduciendo la virulencia de un patógeno pero aun así, manteniéndolo viable ("vivo"). Antes de adentrarnos en los detalles de lo que plantea este trabajo como metodología para racionalizar el diseño de vacunas atenuadas, veremos brevemente algunos antecedentes y cuál fue, y sigue siendo, la metodología clásica para la producción de este tipo de vacunas.

4.1. Diseño clásico

La metodología para la producción de vacunas atenuadas ha sido, históricamente el pasaje del virus a través de cultivos de células distintas a las células huésped. De esta manera, el virus tiende a "evolucionar" para adaptarse al nuevo huésped y ser capaz de reproducirse.

Este concepto de "evolución" se traduce en alguna cantidad de mutaciones sobre la secuencia de nucleótidos del virus, que se espera, reduzcan su capacidad de reproducirse en el huésped original. Luego, son precisamente estas mutaciones las que le confieren la atenuación y dan lugar a la vacuna atenuada.

La principal desventaja en este proceso es que las mutaciones que se producen son totalmente impredecibles, y aún cuando estas mutaciones derivan en un virus atenuado, éste podría revertir muy fácilmente a la virulencia dependiendo de la naturaleza de las mutaciones que generan la atenuación[3]. Además, como vimos en la sección 2.3, la alta frecuencia de mutaciones que poseen los virus RNA aumenta la probabilidad de reversión a la virulencia. De hecho, esto es lo que ocurre con muchas vacunas atenuadas y, en particular, con la OPV.

A pesar de este peligro de reversión a la virulencia, ésta sigue siendo la principal metodología para la producción de vacunas atenuadas. Esto se debe,

fundamentalmente, a la falta de conocimiento sobre el significado o incidencia de las mutaciones en la atenuación de un determinado virus. Sin embargo, los recientes avances en la virología molecular han permitido explorar nuevas técnicas que permitan controlar la replicación de un virus o su virulencia lo que abrió la puerta a lo que se denomina "diseño racional de vacunas atenuadas"[14].

4.2. Diseño racional

La idea central en esta nueva metodología, dentro de la que se enmarca este trabajo, es explotar el conocimiento que se ha producido en los últimos años acerca de la biología molecular de determinados virus. Pudiendo controlar la replicación de un virus o su virulencia, sería posible diseñar vacunas atenuadas "seguras" evitando la impredecibilidad de las atenuaciones empíricas obtenidas mediante el diseño clásico.

4.2.1. Antecedentes

Existen diferentes aproximaciones al diseño racional de vacunas atenuadas[14]. En general, todas estas aproximaciones se encuentran en fase experimental y todavía no se han aplicado en producción. A continuación hacemos mención a tan solo dos de ellas, fundamentalmente para marcar la diferencia con el diseño clásico que presentamos anteriormente.

Fidelidad en la replicación[20]

Como ya vimos en la sección 2.3, la alta frecuencia de mutaciones en los virus RNA se debe a la alta tasa de error en su RNA polimerasa. Luego, modificando la RNA polimerasa de tal manera que se reduzca su tasa de error, se obtendría un virus atenuado mas estable y con menor probabilidad de revertir a la virulencia en las sucesivas replicaciones.

La principal desventaja de esta aproximación radica en que las posibles variantes sobre la RNA polimerasa deben ser determinadas y evaluadas experimentalmente para cada virus en particular.

(De-)Optimización de codones[19, 6]

En la sección 2.1 sobre el código genético, vimos que cada aminoácido puede ser codificado hasta por 6 codones distintos. Es decir, distintas secuencias de nucleótidos resultan equivalentes en términos de los aminoácidos que codifican. Concretamente, una proteína de 300 aminoácidos puede ser codificada por aproximadamente 10^{151} secuencias de nucleótidos.

Sin embargo, experimentalmente se pudo comprobar que algunos codones son más frecuentes que otros (*codon bias*). Similarmente, pero de manera independiente, se comprobó que determinados pares de codones son más frecuentes que otros (*codon pair bias*). Aunque todavía no está claro a qué se debe esta "parcialidad" en el uso de codones o pares de codones, se supone que afectaría el proceso de síntesis de proteínas (traducción).

Lo que se propone con esta aproximación, es determinar las secuencias de nucleótidos que conserven la secuencia aminoacídica del virus pero que, al mismo tiempo, tiendan a usar codones y pares de codones menos frecuentes. De

esta manera, la atenuación del virus se obtendría debilitando su capacidad de traducción y replicación.

Entre las ventajas que presenta esta metodología, se destaca por un lado que la atenuación es el resultado de un análisis sistemático y por lo tanto, aplicable a diferentes virus de manera automática. Por otro lado, la alta cantidad de cambios que se realizan sobre el virus original sugieren una menor probabilidad de revertir a la virulencia.

4.3. Propuesta de solución

En pocas palabras, la propuesta consiste en encontrar un conjunto de secuencias de RNA que conserven las propiedades que le otorgan la atenuación al virus y que, al mismo tiempo, tiendan a maximizar el número de mutaciones necesarias para alcanzar secuencias semejantes a las patógenas o revertantes.

Esto tiene algunos puntos en común con la "(de-)optimización de codones" que presentamos anteriormente. En particular, ambas aproximaciones comparten la idea de sistematizar el diseño de forma tal que pueda ser usado para diferentes virus. Esto implica, fundamentalmente, plasmar la metodología en la implementación de un software que, a partir de una serie de datos provistos por el usuario, devuelva como resultado una o varias secuencias de nucleótidos que representen posibles atenuaciones del virus.

De una forma más abstracta, podemos pensar en un software para el diseño de secuencias de nucleótidos basado en restricciones. Fundamentalmente, lo que se busca es "generar" secuencias que satisfagan determinadas propiedades o restricciones, en particular, aquellas que tiendan a reducir la virulencia del virus.

En este sentido, se podría trazar una analogía con los algoritmos para la predicción inversa de estructura secundaria (*inverse folding*). En esencia, estos algoritmos "diseñan" secuencias de RNA que tengan como estructura secundaria mfe, la estructura dada por el usuario. De hecho, como mencionamos en la sección 3.2.2, el problema se plantea computacionalmente como un CSP.

La principal innovación de esta propuesta es que las restricciones se enfocan esencialmente en el IRES y su estructura secundaria. Por lo visto en la sección 2.3, parece clara la importancia del IRES en la traducción y posterior replicación de diferentes virus RNA y en particular del poliovirus.

Lo que nos proponemos en este trabajo es realizar un análisis sistemático de las posibles variantes al IRES de los virus atenuados Sabin (y eventualmente cualquier otro) que conserven la estructura secundaria y en consecuencia, la atenuación del virus. Luego, maximizando la cantidad de mutaciones necesarias para revertir a secuencias semejantes a las patógenas o revertantes, estaríamos reduciendo la probabilidad de que el virus atenuado sufra reversión a la virulencia.

Esquemáticamente, podemos plantear el problema de la siguiente manera:

- **Entrada:** Genoma del virus atenuado, genomas de los patógenos o revertantes y un conjunto de restricciones. Fundamentalmente, la conservación de la estructura secundaria del IRES del virus atenuado.
- **Objetivo:** Satisfaciendo las restricciones impuestas, maximizar la distancia entre el genoma del virus atenuado y los genomas patógenos o revertantes.

- **Salida:** Una o varias secuencias candidatas a "mejorar" el virus atenuado.

4.3.1. Formalización

Como mencionamos en la sección 1.5 el problema puede ser visto como un problema de **"optimización combinatoria basado en restricciones"**. Pero para hacerlo, primero se deben identificar algunos elementos fundamentales que permitan definir el problema.

Este tipo de problemas consiste en asignar valores a un conjunto finito de variables (componentes de una solución) que satisfagan determinadas restricciones. En nuestro caso, estas restricciones serán propiedades biológicas sobre partes de una secuencia de RNA, y los posibles valores a asignar serán (sub)secuencias de RNA que satisfagan las propiedades requeridas.

Sobre diferentes partes de la secuencia de RNA se pueden requerir diferentes propiedades biológicas (restricciones). Luego, serán estas propiedades las que determinen los posibles valores sobre cada parte "variable" de la secuencia. Finalmente, las combinaciones de los posibles valores para cada parte de la secuencia, formarán las potenciales soluciones del problema.

Definición del problema

Sea N la longitud de la secuencia de RNA del virus atenuado, y para $k \in \mathbb{N}$ sea $\mathcal{S}_k$ el conjunto de secuencias de RNA de longitud k. Entonces definimos:

- **Espacio de soluciones:** $\mathcal{S}_N$
- **Componentes variables de una solución:** $s_1, s_2, \ldots, s_n$ tal que $s_i \in \mathcal{S}_{N_i}$ con $1 \leq i \leq n$ y $0 < N_i \leq N$.
- **Restricciones sobre las componentes:** Conservación de la estructura secundaria o de la secuencia aminoacídica con respecto al virus atenuado. Eventualmente, se podrían contemplar otras restricciones que impliquen propiedades biológicas que resulten de interés para la atenuación del virus.
- **Función "objetivo" o de evaluación:** $f : \mathcal{S}_N \to \mathbb{R}$ tal que $f(s)$ calcula la bondad de cada solución, en nuestro caso, como la distancia en número de mutaciones necesarias para llegar de s a alguna secuencia patógena o revertante.

Lo primero que podemos mencionar es que para cualquier virus RNA, recorrer el espacio de soluciones de manera exhaustiva es inviable ya que, por ejemplo, para el caso del poliovirus $N \simeq 7,500$. Es decir que existen aproximadamente $4^{7,500}$ posibles secuencias de RNA (7,500 posiciones y 4 bases de nucleótidos posibles para cada posición).

Por otro lado, la posibilidad de evaluar los posibles valores para cada componente s_i de manera exhaustiva, dependerá fundamentalmente del tipo de restricción impuesta sobre esa componente y de la longitud N_i. En particular para los virus atenuados Sabin y la conservación de la estructura secundaria del IRES ($N_i \simeq 400$), la evaluación exhaustiva es inviable debido al alto costo de predecir la estructura secundaria, como vimos en las secciones 3.2.1 y 3.2.2.

Con el problema planteado de esta manera y debido a la inviabilidad de realizar una búsqueda exhaustiva, se pueden utilizar diferentes algoritmos de

búsqueda local para que, partiendo de una secuencia de RNA inicial, (en este caso el virus atenuado) recorrer el espacio de búsqueda $\mathcal{S}_N$ teniendo como objetivo maximizar la función de evaluación f.

En este contexto, la definición de la función de evaluación $f : \mathcal{S}_N \to \mathbb{R}$ es crucial para encontrar buenas soluciones. En principio se podría pensar que la imagen de la función sea $\mathbb{N}$ en lugar de $\mathbb{R}$. De hecho, éste es el caso si la función calcula la distancia de *Hamming* estándar, esto es, sumar 1 por cada una de las bases en las que difiere la secuencia solución de la secuencia patógena. Pero de esta manera se estaría suponiendo que la probabilidad de mutación entre las bases es uniforme y esto no es así necesariamente.

Luego, determinando empíricamente la probabilidad de mutación de cada base hacia cualquiera de las otras tres, se podría incluir esta información en la función de evaluación usando una matriz de dimensión 4×4 indicando en cada posición (i, j), el costo de realizar la mutación de la base i a la base j. En particular, para la distancia de *Hamming* podemos definir la siguiente matriz de costos:

$$M = \begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{Bmatrix}$$

Parte III

Desarrollo del Software

Capítulo 5

Proceso de desarrollo

> How does a project get to be a year late?... One day at a time.
>
> Fred Brooks

A esta altura ya hemos presentado los principales conceptos biológicos involucrados en este trabajo, el diseño de vacunas atenuadas y la formalización del problema que nos propusimos resolver. En los capítulos restantes veremos los aspectos técnicos más relevantes sobre el desarrollo del software.

5.1. Modelo de desarrollo

Para llevar adelante el desarrollo del software, se optó por el clásico modelo de cascada. Fundamentalmente debido a su simplicidad y a que los requerimientos con que debía cumplir el software estaban bien definidos desde el inicio, no solo a nivel funcional, sino también en cuanto a principios del diseño orientado a objetos.

5.1.1. Etapas de la cascada

Las etapas que se llevaron a cabo durante el desarrollo de este trabajo y que veremos con mayor detenimiento en los siguientes capítulos, fueron las siguientes:

1. Especificación de requerimientos.
2. Diseño.
3. Implementación.
4. Verificación.

Quedaron fuera del alcance las etapas “Instalación” y, por supuesto, “Mantenimiento”.

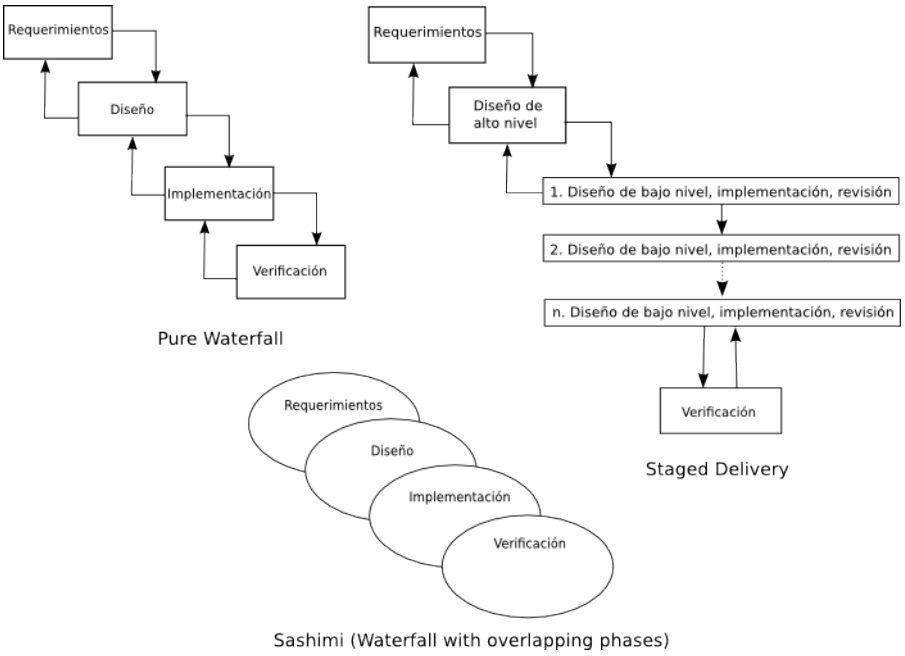

Figura 5.1: Modelos de desarrollo

5.1.2. Consideraciones del modelo

Para implementar el modelo de cascada se tuvieron en cuenta algunas consideraciones con respecto al contexto en el que se desarrolló el software y que veremos a continuación.

Por un lado, todas las etapas que enumeramos más arriba debían ser llevadas a cabo por la misma persona. Esto es significativamente distinto a contar con grupos de personas o equipos independientes para cada etapa y en donde la documentación generada en cada fase es fundamental para la comunicación entre los distintos equipos. En este sentido, usamos un modelo de cascada con "solapamiento", también conocido como *Sashimi*[17], que permite empezar una etapa sin haber terminado por completo la anterior. Debido a que en este modelo es más natural la continuidad del personal entre las distintas etapas, no es necesario generar tanta documentación como en el modelo de cascada "puro" y esto agilizó el proceso de desarrollo.

Por otro lado, el rol de "cliente" en el proceso de desarrollo fue cubierto por miembros de FuDePAN que garantizaron el conocimiento del dominio de manera tal de poder realizar la "Especificación de requerimientos" pero al mismo tiempo, contaban con los conocimientos técnicos de diseño y programación orientada a objetos como para supervisar el desarrollo en estos aspectos. En este sentido, se tomaron algunos elementos de la variante al modelo de cascada "puro" que se conoce como *Staged Delivery*[17] o "implementación incremental" que nos permitieron realizar revisiones periódicas con miembros de FuDePAN durante las etapas de "Diseño" e "Implementación".

En la Figura 5.1 se pueden ver el modelo de cascada "puro" y las variantes descriptas anteriormente y que adoptamos para este trabajo.

Por último, con el fin de validar los requerimientos del software que se consideraban más importantes, durante la etapa de "Diseño", se implementó un prototipo utilizando el lenguaje de programación Python.

5.2. Ecosistema

El "ecosistema" de herramientas que se utilizaron para llevar adelante el proceso de desarrollo son las siguientes:

- **Lenguaje de programación:** El software se implementó en **C++**[1].
- **Lenguaje de diseño:** El diseño del software se hizo utilizando **UML**[2].
- **Control de versiones:** Se utilizó el sistema de control de versiones SVN[3] y puede ser consultado en `http://vac-o.googlecode.com`.
- **Sistema de "construcción":** Para automatizar el proceso de compilación del código fuente se utilizó **CMake**[4].
- **Automatización de pruebas:** Para realizar la verificación del software, se optó por implementar pruebas unitarias utilizando **google-test**[5] y **google-mock**[6].
- **Análisis estático:** Se utilizaron las herramientas **astyle**[7] y **cppcheck**[8].
- **Análisis dinámico:** Se utilizaron las herramientas **valgrind**[9] y **gcov**[10] para verificar la ausencia de "memory leaks" y la cobertura de las pruebas.

[1] `http://cplusplus.com`
[2] `http://www.uml.org`
[3] `http://subversion.apache.org`
[4] `http://www.cmake.org`
[5] `http://googletest.googlecode.com`
[6] `http://googlemock.googlecode.com`
[7] `http://astyle.sourceforge.net`
[8] `http://sourceforge.net/apps/mediawiki/cppcheck`
[9] `http://valgrind.org`
[10] `http://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

Capítulo 6

Requerimientos

> If you think it's simple, then you have misunderstood the problem.
>
> Bjarne Stroustrup

Los requerimientos del software fueron provistos por miembros de FuDePAN y quedaron documentados en la "Especificación de Requerimientos de Software" que se anexa a este trabajo. A continuación haremos mención a los requerimientos más relevantes y que nos permitan introducir los principales aspectos del software.

6.1. Descripción general

Retomando lo dicho en la sección 4.3, podríamos resumir los requerimientos funcionales de vac-o de la siguiente manera:

- **Entrada:** Genoma del virus atenuado, genoma del patógeno o revertantes y un conjunto de restricciones sobre partes de la secuencias del virus atenuado.
- **Objetivo:** Satisfaciendo las restricciones impuestas, maximizar la cantidad de mutaciones necesarias para que el virus atenuado se convierta en una secuencia patógena o revertante.
- **Salida:** Una o varias secuencias candidatas a "mejorar" el virus atenuado.

Desde el punto de vista del diseño, el sistema debía ser altamente modular, de forma que sea apto para encontrar secuencias genómicas para diferentes tipos de organismos. En este sentido, se debieron seguir los principios de diseño conocidos por el acrónimo **SOLID**[16].

- *Single Responsability Principle* (SRP)
- *Open-Close Principle* (OCP)
- *Liskov Substitution Principle* (LSP)

- *Interface Segregation Principle* (ISP)
- *Dependency Inversion Principle* (DIP)

6.2. Extensiones

Uno de los principales requerimientos con que debía cumplir el software es que sea versátil, es decir, que permita ser configurado y extendido de una forma simple y sin necesidad de modificar sus componentes centrales.

En este sentido, se propuso desde un comienzo un software que funcionaría en base a la información provista por una extensión o *plugin*. Básicamente, la responsabilidad de una extensión sería la de proveer a vac-o la información relevante para cada virus atenuado que se desee optimizar. En particular, el genoma del virus atenuado, el genoma del patógeno y el conjunto de restricciones que se deben satisfacer durante la búsqueda. Las extensiones también serían responsables de evaluar las secuencias candidatas y de determinar la estrategia de búsqueda.

6.3. Regiones combinatorias

Como ya mencionamos anteriormente, el problema se puede ver como un problema de "optimización combinatoria basado en restricciones". Luego, un ingrediente que no podemos dejar de mencionar son, precisamente, las restricciones.

Sobre una secuencia de RNA se distinguen dos tipos de regiones, ORF y UTR. Estas regiones de la secuencia o eventualmente ciertas partes de interés, tienen diferentes propiedades o funciones biológicas (la estructura secundaria de RNA o los aminoácidos que codifican) que permiten considerar a la secuencia de RNA como perteneciente a un virus atenuado (reducen la virulencia del patógeno manteniéndolo viable o "vivo").

Sobre las diferentes partes de interés (para la atenuación del virus) de una secuencia de RNA, que llamaremos "regiones combinatorias", se aplicarán restricciones que tiendan a mantener las propiedades o funciones biológicas que garantizan la atenuación del virus.

En este contexto nos referimos a las "variantes" de una región combinatoria como los posibles valores ((sub)secuencias de RNA) que satisfacen la restricción impuesta por la región combinatoria. Así, podemos definir a una región combinatoria de la siguiente manera:

Definición 3. Dada una secuencia de RNA S de longitud n, una región combinatoria sobre S será una 4-upla $(inicio, fin, tipo, eval)$ tal que:

- $0 \leq inicio < fin \leq n$
- $tipo$ determina la restricción sobre la región y, por ende, queda definido un conjunto $\mathcal{V}$ de posibles variantes a la región.
- $eval : \mathcal{V} \rightarrow (0, 1)$ es una función de evaluación local a la región que determina la bondad de una variante determinada.

Si suponemos $N \geq 1$ y $R_1, \ldots, R_N$ regiones combinatorias, tendremos que el número de posibles secuencias que vac-o "debería" evaluar, sera igual al cardinal del producto cartesiano $\mathcal{V}_1 \times \cdots \times \mathcal{V}_N$ con $\mathcal{V}_1, \ldots, \mathcal{V}_N$ variantes de cada región combinatoria respectivamente. Con el objetivo de que las secuencias evaluadas cumplan con un mínimo criterio de "calidad" y al mismo tiempo restringir el número de posibilidades, solo serán tenidas en cuenta aquellas combinaciones de las variantes a cada región combinatoria tal que el producto de sus respectivas evaluaciones locales (*eval*) sea mayor o igual a un "umbral" determinado por la extensión. Es decir, serán tenidas en cuenta las combinaciones de variantes:

$$(v_1, \ldots, v_N) \in \mathcal{V}_1 \times \cdots \times \mathcal{V}_N \text{ tal que } \prod_{i=1}^{N} eval(v_i) \geq umbral$$

Para el caso testigo de la OPV, se contemplaron dos tipos de regiones combinatorias aunque se debe tener en cuenta la posibilidad de que en el futuro se agreguen mas. Estos tipos de regiones combinatorias o restricciones son:

- Conservación de la estructura secundaria de RNA.
- Conservación de la secuencia de aminoácidos.

Además, para el caso de la "conservación de estructura secundaria de RNA" se debía contar con la posibilidad de utilizar diferentes algoritmos de predicción directa e inversa de manera "transparente".

6.4. Estrategias de búsqueda

Otro de los aspectos "configurables" del software debía ser la estrategia de búsqueda utilizada. En este sentido, la extensión es responsable de proveer al sistema la "forma" de recorrer el espacio de búsqueda y al mismo tiempo determinar cuando se debe terminar el recorrido.

Vale aclarar que todos los algoritmos de búsqueda local son "incompletos" en el sentido de que no recorren exhaustivamente el espacio de búsqueda como sí lo hacen los algoritmos mas tradicionales o sistemáticos. Luego, es necesario contar con un criterio de terminación que suele estar relacionado con la cantidad de iteraciones realizadas, la "calidad" de las soluciones encontradas o la cantidad de iteraciones sin que se produzcan mejores soluciones.

6.5. Control de calidad

Cada secuencia encontrada por vac-o durante el recorrido del espacio de búsqueda, debe ser sometida a un "control de calidad" complementario al requerimiento sobre la calidad de las variantes a cada región combinatoria que mencionamos anteriormente ($\prod_{i=1}^{N} eval(v_i) \geq umbral$).

Básicamente se pretende simular las mutaciones acumuladas que se producen en las sucesivas replicaciones del virus en la naturaleza y sobre cada secuencia mutante, verificar determinadas propiedades que brinden mayor seguridad sobre la atenuación del virus.

Análogamente a las regiones combinatorias, definimos las regiones de validación de la siguiente manera.

Definición 4. Dada una secuencia de RNA S de longitud n, una región de validación sobre S será una 5-upla $(inicio, fin, prueba, criterio, prof)$ tal que:

- $0 \leq inicio < fin \leq n$
- *prueba* determina la forma en que se producen las mutaciones.
- *criterio* determina la propiedad que deben cumplir todas las secuencias mutantes generadas.
- $prof > 0$ determina la profundidad con que se realiza el control de validación (ciclos de replicación que se simulan).

Se puede pensar al control de calidad como la generación de un árbol de secuencias. La raíz del árbol será la secuencia candidata y los nodos del árbol serán las secuencias mutantes generadas mediante la forma seleccionada en la variable *prueba* y que cumplan con la propiedad determinada por la variable *criterio*. Si una secuencia mutante no cumple con esa propiedad, no se genera ese nodo del árbol, se corta en ese punto el control de calidad y la secuencia candidata no pasa el control de calidad sobre esa región de validación.

Diremos que una secuencia pasa el control de calidad sobre una región de validación cuando sea posible generar todos los nodos del árbol hasta alcanzar la profundidad $prof$. Luego, solo serán consideradas aquellas secuencias candidatas que pasen el control de calidad sobre todas las regiones de validación.

Para las formas de producir mutaciones acumuladas se contemplaron "mutaciones sistemáticas" y "mutaciones aleatorias". Por otro lado, las propiedades que se contemplaron como criterio de verificación son "similitud estructural" y "disimilitud estructural". Por similitud y disimilitud estructural nos referimos a la comparación entre la estructura secundaria de RNA de cada secuencia mutante y una estructura secundaria de RNA dada por la extensión.

Si suponemos $\mathcal{E}$ el conjunto de estructuras secundarias sobre secuencias de longitud n y una función $StructureCompare : \mathcal{E} \times \mathcal{E} \to (-\infty, 1)$ tal que dadas $e_1, e_2 \in \mathcal{E}$ calcula la similitud entre ellas, siendo 1 el valor de similitud máxima ($e_1 = e_2$), entonces, para $s \in (-\infty, 1)$ tenemos que:

- Similitud estructural: $StructureCompare(e_1, e_2) \geq s$
- Disimilitud estructural: $StructureCompare(e_1, e_2) \leq s$

De todas formas, se debía contemplar la posibilidad de que en el futuro se agreguen otras formas de generar mutaciones y otros criterios de verificación.

6.6. Ranking de secuencias candidatas

Finalmente, nos queda por describir el ranking de secuencias candidatas y que representa fundamentalmente el resultado de la ejecución de vac-o. Esto es simplemente un listado de secuencias de RNA, ordenadas según la evaluación que hiciera la extensión. Básicamente, las secuencias mejor evaluadas serán aquellas que estén a mayor distancia (para alguna definición de distancia) de la secuencia patógena o revertante y, por ende, tendrán menor probabilidad de sufrir reversión a la virulencia.

Capítulo 7

Diseño

> The effective exploitation of his powers of abstraction must be regarded as one of the most vital activities of a competent programmer.
>
> Edsger W. Dijkstra

Al igual que con los requerimientos del software, la descripción detallada del diseño quedó documentada en la "Especificación de Diseño de Software" y que se agrega como anexo. Análogamente al capitulo anterior, nos limitaremos a mencionar sólo los aspectos más relevantes del diseño, dejando como lectura adicional el documento anexo.

7.1. Metodología

La metodología adoptada para realizar el análisis y descripción del diseño se denomina *Responsibility-Driven Design* (RDD)[21]. Esta técnica de diseño orientado a objetos se enfoca en qué responsabilidades deben ser cubiertas por el sistema y en cuáles serán los objetos responsables de llevarlas a cabo. Por "responsabilidades" se entiende fundamentalmente lo siguiente:

- Las acciones que realiza un objeto.
- El conocimiento o la información que mantiene un objeto.
- Las decisiones que realiza un objeto afectando a otros.

Objetivos del diseño

Uno de los requerimientos del software fue que se respetaran los principios del diseño orientado a objetos resumidos en el acrónimo **SOLID**[16]. En particular se puso especial atención en respetar los principios SRP, OCP y DIP debido a que repercuten directamente en que el sistema sea más simple de mantener y extender con nuevas funcionalidades. Además, el principio DIP es fundamental para lograr un software que sea verificable mediante la automatización de pruebas como se pudo comprobar más adelante, en la etapa de "verificación".

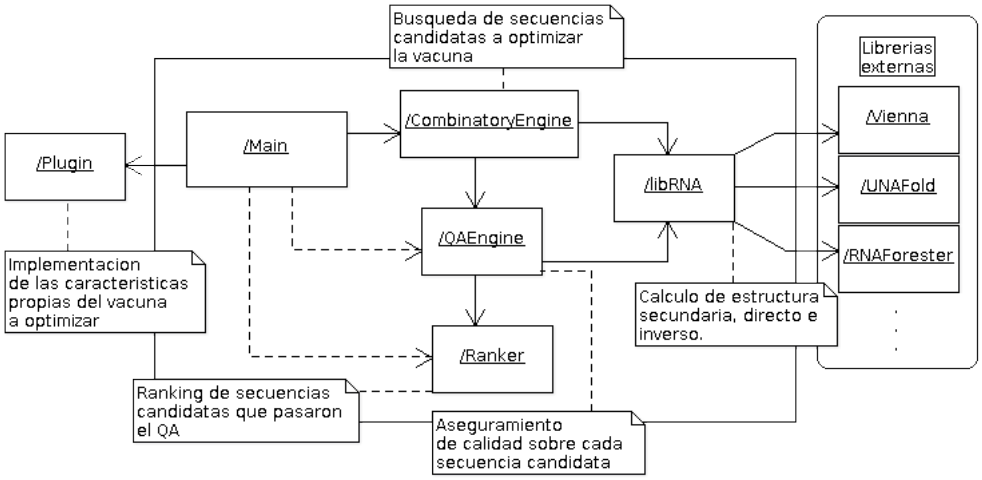

Figura 7.1: Arquitectura de vac-o.

7.2. Arquitectura

A continuación veremos los principales componentes del sistema y siguiendo la metodología adoptada, sus respectivas responsabilidades. En las Figuras 7.1 y 7.2 se pueden ver los diagramas UML de la arquitectura y de flujo respectivamente.

- **Main:** Representa el componente principal en términos de ejecución del sistema. Comprende principalmente, la inicialización y configuración de otros componentes.
- **Plugin:** Representa la implementación de las características propias de la vacuna que se desea optimizar. Brinda información para la configuración inicial como así también, los criterios para evaluar las secuencias candidatas.
- **CombinatoryEngine:** Representa el motor combinatorio del sistema encargado de encontrar nuevas secuencias que sean candidatas a optimizar la atenuación del virus.
- **QAEngine:** Representa el motor de control de calidad del sistema encargado de decidir si una secuencia candidata pasa el control o no.
- **Ranker:** Representa el componente encargado de mantener un "ranking" de secuencias que será además el resultado final de la ejecución.
- **libRNA:** Representa el componente que provee al sistema funcionalidades para la manipulación de secuencias y estructuras de RNA ("folding" directo e inverso y comparación estructural, entre otras) utilizando librerías externas.

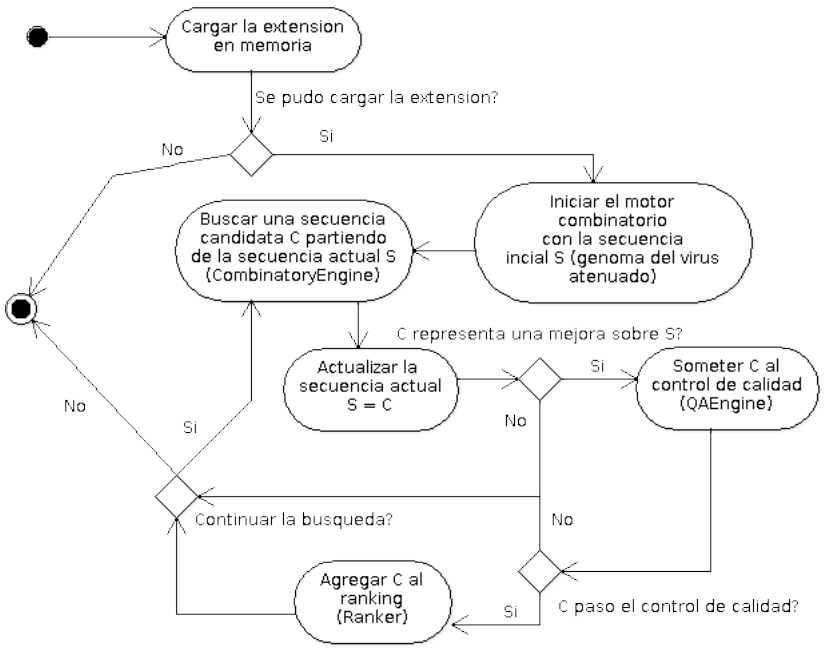

Figura 7.2: Diagrama de flujo de vac-o.

En las siguientes secciones profundizaremos sobre los componentes "libRNA" y "CombinatoryEngine" por ser probablemente los más importantes en cuanto a sus responsabilidades y dejaremos como lectura adicional la "Especificación de Diseño de Software" que contiene una descripción detallada de todos los componentes de la arquitectura presentada. Al mismo tiempo, profundizar sobre estos componentes nos permitirá introducir los principales patrones de diseño que se utilizaron de forma análoga en otras partes del sistema. Entre otros, se destacan *Iterator*, *Observer*, *Template Method* y *Visitor*. Para una descripción de estos y otros patrones del diseño orientado a objetos, ver [8].

7.3. Librerías externas

Como vimos en la secciones 3.2.1 y 3.2.2, existen diversas implementaciones que resuelven el problema de la predicción (directa e inversa) de estructura secundaria de RNA y lo mismo ocurre para el caso de la comparación estructural. Además no se descarta la posibilidad de que en el futuro aparezcan nuevas y mejores implementaciones.

Por todo esto, uno de los requerimientos del software fue que sea posible el uso de cualquiera de estas implementaciones de manera transparente. El problema radica en que cada librería tiene diferentes formas de recibir los parámetros de entrada y diferentes formas de dar los resultados que genera. A raíz de esto se propuso el componente "libRNA" como una forma de unificar el acceso a estas librerías externas e integrarlas al resto del sistema. Las interfaces propuestas fueron las siguientes:

- **IFold:** Provee la predicción directa (*folding*) de secuencias de RNA.
- **IFoldInverse:** Provee la predicción inversa (*inverse folding*) de secuencias de RNA.
- **IStructureCmp:** Provee la comparación de estructuras secundarias de RNA.
- **ISequenceCmp:** Provee la comparación de secuencias de RNA.

En esto podemos ver la idea del principio de diseño DIP. Haciendo que el sistema dependa de estas interfaces en lugar de sus respectivas implementaciones conseguimos abstraer los detalles de cada librería externa y logramos un software mas versátil. En el capitulo 8 veremos algunos detalles sobre las implementaciones de estas interfaces y en particular de **IFoldInverse**.

7.4. Motor combinatorio

El componente "CombinatoryEngine" contiene todo lo referente al recorrido del espacio de búsqueda por lo que es, claramente, uno de los componentes principales de vac-o, esto es fundamentalmente, las regiones combinatorias y las diferentes estrategias de búsqueda.

Para permitir que el software sea de utilidad para diferentes tipos de virus, tanto las regiones combinatorias como la estrategia de búsqueda utilizada para la optimización, debían ser altamente configurables. Luego, el objetivo sobre este componente fue capturar la estructura general de los algoritmos de búsqueda local que suelen denominarse de "mejoramiento iterativo". Algunos de estos algoritmos son:

- *First Improvement*
- *Best Improvement*
- *Simulated Annealing*
- *Tabu Search*

En todos estos algoritmos, y en general en la búsqueda local, están presentes los conceptos de "vecindario" (*neighborhood*) y de "movimiento" (*step*) que veremos con mayor detenimiento en el capitulo 9. Mientras tanto, la siguiente introducción nos servirá para justificar las interfaces propuestas.

Si el espacio de búsqueda es $\mathcal{S}$, entonces un "vecindario" será una relación $\mathcal{N} \subseteq \mathcal{S} \times \mathcal{S}$ y el "movimiento" (entre soluciones en el espacio de búsqueda) sera una función $step : \mathcal{S} \to \mathcal{D}(\mathcal{S})$. Donde $\mathcal{D}(S)$ denota el conjunto de distribuciones de probabilidad sobre un conjunto dado S y donde una distribución de probabilidad $D \in \mathcal{D}(S)$ es una función $D : S \to \mathbb{R}_0^+$ que mapea los elementos de S a sus respectivas probabilidades.

Luego, en cada iteración de la búsqueda hay dos responsabilidades bien diferenciadas e independientes. Por un lado, se debe explorar el "vecindario" de la solución actual. Es decir, si la solución actual es s y el "vecindario" es $\mathcal{N}$, debemos generar el conjunto de soluciones $\mathcal{N}(s)$. Por otro lado, una vez que se tiene el conjunto $\mathcal{N}(s)$ de "vecinos" de s, se debe seleccionar uno de ellos utilizando

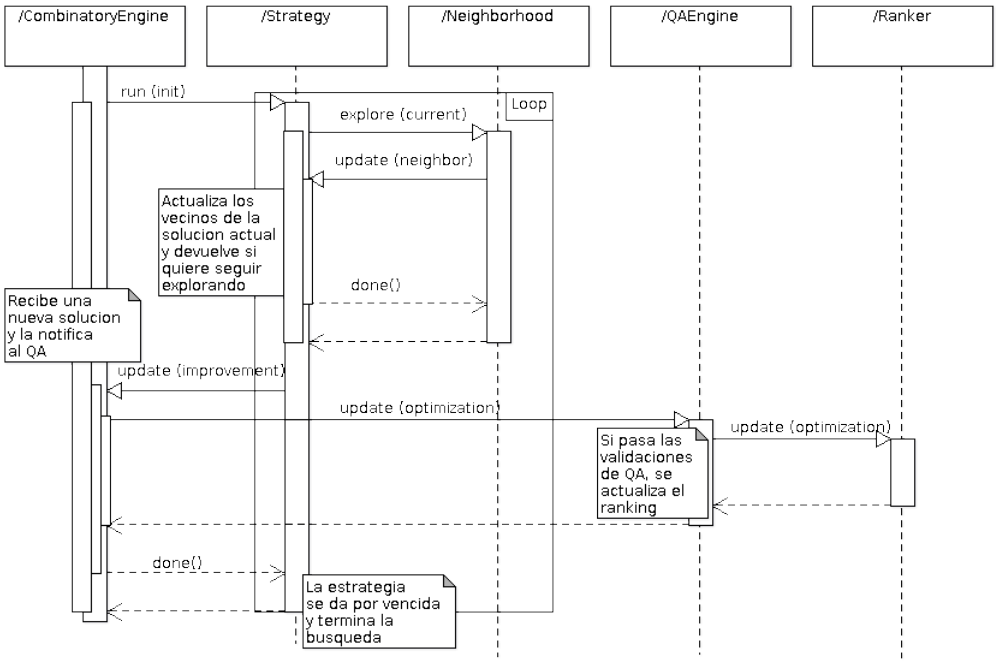

Figura 7.3: Motor combinatorio de vac-o.

la función *step*. Notar que no siempre la solución seleccionada sera mejor que la solución actual. Inclusive los algoritmos que permiten los llamados, "movimientos malos" (seleccionar con una probabilidad p una solución peor que la actual) suelen conducir a mejores resultados.

En este sentido se propusieron la siguientes interfaces:

- **ICombinatoryRegion:** Genera las variantes de la región combinatoria que satisfagan la restricción asociada.
- **ISolution:** Representa una solución en el espacio de búsqueda.
- **INeighborhood:** Explora el vecindario de una solución.
- **IStrategy:** Decide como pasar de una solución a la siguiente y notifica las soluciones que "mejoran" a la solución anterior.
- **ICombinatoryEngine:** Inicia la búsqueda y notifica a otros componentes del sistema las sucesivas soluciones encontradas.

Para terminar de entender la interacción entre estas interfaces, en la Figura 7.3 se puede ver el diagrama de secuencia del motor combinatorio.

El principal patrón de diseño que utilizamos en este componente es el que se denomina *Observer* y que sirve fundamentalmente para lograr la comunicación entre diferentes objetos sin que se produzca acoplamiento. La idea es establecer

una relación entre un emisor y uno o varios receptores sin necesidad de generar una dependencia entre el objeto emisor y el objeto receptor. Además para permitir el uso del patrón *Observer* entre diferentes tipos de objetos, se propuso el patrón de diseño *Template Method*.

En nuestro caso, **INeighborhood** notifica a **IStrategy** por cada solución que se genera mientras se explora el vecindario de la solución actual. Posteriormente, si la solución seleccionada es mejor que la solución actual, **IStrategy** notifica a **ICombinatoryEngine** la solución seleccionada y éste a su vez, la notifica a otros componentes de vac-o (control de calidad).

Una vez más, remarcamos que las dependencias son entre interfaces y no entre implementaciones como establece el principio de diseño DIP. Esto nos brinda la versatilidad de definir diferentes tipos de regiones combinatorias y estrategias de búsqueda sin modificar en absoluto la implementación del motor combinatorio.

Capítulo 8

Integración de librerías externas

> In theory, there is no difference between theory and practice. But, in practice, there is.
>
> Jan L. A. van de Snepscheut

A continuación veremos los detalles de implementación más relevantes de las interfaces propuestas en la sección 7.3 y en particular para los algoritmos de predicción inversa de la estructura secundaria de RNA.

8.1. Implementar vs. Integrar

Ante el requerimiento de contar con algoritmos para realizar la comparación y predicción (directa e inversa) de la estructura secundaria de RNA, debimos evaluar y decidir entre implementar completamente los algoritmos o integrar/adaptar los algoritmos existentes.

En este sentido, las interfaces propuestas en la sección 7.3 no descartan ninguna de las dos posibilidades, sino simplemente especifican los "servicios" que deben ofrecer al sistema las eventuales implementaciones (propias o ajenas).

Por otro lado, implementar este tipo de algoritmos requiere de un profundo conocimiento del dominio, en este caso de química molecular, que se desviaba de los objetivos del software y del trabajo en general. Además, debido a la complejidad de los mismos, se requieren validaciones empíricas para demostrar la fidelidad de los resultados lo cual excedía el alcance de este trabajo.

Finalmente se optó por integrar al sistema los siguientes algoritmos para implementar las respectivas interfaces:

- **IFold:** RNAfold[11].
- **IFoldInverse:** RNAinverse[11], INFO-RNA[4].
- **IStructureCmp:** RNAforester[10]

Nótese que no incluimos la interfaz **ISequenceCmp** debido a que la comparación entre secuencias de RNA es significativamente más simple y para este caso, se optó por realizar una implementación propia utilizando una matriz de costos como se propuso en la sección 4.3.1.

8.2. Integración

La integración de los programas RNAfold y RNAforester no presentaron mayores dificultades ya que en ambos casos la implementación consiste simplemente en ejecutar el programa y luego leer el resultado. Los dos programas realizan cálculos complejos y costosos computacionalmente pero determinísticos. Por otro lado, los programas que realizan la predicción inversa de la estructura secundaria de RNA o *inverse folding* (INFO-RNA y RNAinverse) presentan algunas complicaciones principalmente debido a su no determinismo y que veremos a continuación.

8.2.1. Predicción inversa (*inverse folding*)

Recordando lo dicho en la sección 3.2.2, el objetivo de los programas (INFO-RNA y RNAinverse) que realizan el *inverse folding* es encontrar secuencias de RNA que tengan la estructura secundaria de RNA dada por el usuario. El problema surge cuando se los quiere usar para explorar de forma sistemática, y posiblemente exhaustiva, el espacio de todas las posibles secuencias de RNA que tengan la estructura secundaria dada, como es nuestro caso. Es decir, lo que pretendemos al usar estos programas es tener la capacidad de analizar sistemáticamente la mayor cantidad posible de secuencias de RNA que tengan una estructura secundaria de RNA determinada para luego evaluar cuáles de esas secuencias contribuyen de mejor manera a la atenuación del virus.

Caracterización de los algoritmos para la predicción inversa

Más allá de los diferentes parámetros que aceptan cada uno de estos programas, ambos requieren dos parámetros fundamentales. Por un lado, la estructura secundaria de RNA sobre la que se quiere hacer la predicción inversa y por otro, una secuencia de RNA "incompleta" (algunas bases indefinidas) que se usa como secuencia de inicio en el algoritmo de búsqueda. Por lo tanto, no se realiza una búsqueda exhaustiva en todas las posibles secuencias de RNA, sino que solamente se busca una secuencia que cumpla con los requisitos y que esté en la proximidad de esta secuencia de inicio.

La secuencia "incompleta" que se da como parámetro, cumple además el rol de especificar las restricciones que se deben satisfacer durante la búsqueda (qué bases de nucleótidos *deben* estar presentes en determinadas posiciones). Esto por un lado restringe el espacio de búsqueda, pero por otro lado se podrían estar especificando restricciones que no se pueden satisfacer para la estructura secundaria de RNA dada. En nuestro caso de uso, vac-o comienza su ejecución con una secuencia de RNA (el genoma del virus atenuado) que efectivamente tiene la estructura secundaria de RNA que se quiere conservar. Por lo tanto, esta misma secuencia se puede usar para generar distintas secuencias "incompletas" con la garantía de que todas especifiquen restricciones satisfactibles.

El no determinismo está dado por el hecho de que para una misma estructura secundaria de RNA y una misma secuencia (incompleta) de inicio, en sucesivas ejecuciones de los programas se pueden encontrar distintas secuencias de RNA aunque eventualmente también repetidas. Por otro lado, modificar la secuencia (incompleta) de inicio podría aunque no necesariamente implicar que se encuentren nuevas secuencias.

Con todo lo dicho, deberían quedar claras las complicaciones que presentan estos programas para ser ejecutados de manera sistemática. Pero para dejarlo del todo claro, podemos formalizarlo de la siguiente manera.

Recordemos que una secuencia de RNA se representa como una palabra sobre el alfabeto $\Sigma = \{A, U, G, C\}$ y una secuencia de RNA "incompleta" como una palabra sobre el alfabeto $\Sigma \cup \{N\}$ donde las N representan las bases indefinidas. Además, si s es una secuencia de RNA (completa o incompleta) de longitud n, s_i será la posición o base i-esima de la secuencia s con $0 \leq i < n$.

Luego, si definimos:

- $\mathcal{S}$ el conjunto de secuencias de RNA de longitud n.
- $\mathcal{E}$ el conjunto de estructuras secundarias de RNA sobre secuencias de longitud n.
- $\mathcal{I}(e) \subseteq \mathcal{S}$ el conjunto de secuencias de RNA que tienen como estructura secundaria a $e \in \mathcal{E}$.
- $\mathcal{R}$ el conjunto de secuencias de RNA incompletas de longitud n.

podemos caracterizar a los algoritmos para la predicción inversa de la estructura secundaria de RNA como $inverse : \mathcal{E} \times \mathcal{R} \to \mathcal{S}$ tal que para $e \in \mathcal{E}$ y $r \in \mathcal{R}$, si $inverse(e, r) = s$ entonces se satisfacen:

- $s \in \mathcal{I}(e)$
- $\{\forall i : 0 \leq i < n \wedge r_i \neq N : s_i = r_i\}$

El problema es básicamente que no conocemos la cardinalidad de $\mathcal{I}(e)$ y que además, si pensamos a las sucesivas ejecuciones de $inverse(e, r)$ como una forma de enumerar el conjunto $\mathcal{I}(e)$, el orden en que se enumeran las secuencias sería aleatorio y permitiría repeticiones (no determinístico).

Propuesta para sistematizar la predicción inversa

Vistas las complicaciones para lograr analizar sistemáticamente todas, o al menos un gran número de secuencias que tengan una estructura secundaria de RNA determinada, veremos de qué manera integramos al sistema los algoritmos mencionados.

La idea consiste básicamente en "recordar" las secuencias encontradas en ejecuciones anteriores y en sistematizar la forma de generar secuencias incompletas. Tener "memoria" sobre las secuencias encontradas es fundamental para no volver a evaluar la misma secuencia. Por otro lado, sistematizar la generación de secuencias incompletas tiende a restringir el espacio de búsqueda desde diferentes "frentes" con el objetivo de alcanzar a cubrir la mayor cantidad del espacio de búsqueda posible.

Para generar secuencias incompletas se propuso recorrer iterativamente todas las formas de seleccionar k posiciones de n, donde n es la longitud de la secuencia completa y k es un parámetro determinado por la extensión. Es decir que tendremos $\binom{n}{k}$ secuencias incompletas distintas. Sin embargo, aún no tenemos forma de determinar cuándo se debería pasar a la siguiente secuencia de inicio ya que el hecho de que el algoritmo de predicción inversa devuelva una secuencia que ya había sido encontrada previamente, no implica que en sucesivas ejecuciones y sin cambiar la secuencia de inicio, se encuentren nuevas secuencias. Luego, se propuso delegar esta responsabilidad a la extensión y que sea ésta quien determine el número m de intentos que se realizan sobre cada secuencia de inicio hasta encontrar una secuencia nueva.

A continuación presentamos el pseudo código de la propuesta asumiendo las siguientes variables:

- $e \in \mathcal{E}$: la estructura secundaria de RNA sobre la que se realiza la predicción inversa.
- $i \in \mathcal{I}(e)$: la secuencia de RNA a la que se le desean buscar variantes que conserven la estructura secundaria e.
- $n \in \mathbb{N}$: la longitud de la secuencia i.
- $r \in \mathcal{R}$: la secuencia incompleta de inicio.
- $k \in \mathbb{N}$: el número de posiciones indefinidas para generar secuencias incompletas.
- $m \in \mathbb{N}$: el número de intentos por cada secuencia incompleta de inicio.
- $f \subseteq \mathcal{I}(e)$: el conjunto de secuencias encontradas.

y los siguientes procedimientos auxiliares:

- $inverse(e, r)$: algoritmo externo como se caracterizó anteriormente.
- $begin(i, n, k)$: devuelve la primera de las $\binom{n}{k}$ secuencias incompletas a partir de la secuencia i.
- $next(r)$: devuelve la siguiente secuencia incompleta, para algún orden de las $\binom{n}{k}$ secuencias incompletas a partir de la secuencia i.

Procedimiento 1 Inicialización

1: $f \leftarrow \emptyset$
2: $r \leftarrow begin(i, n, k)$

Procedimiento 2 Predicción inversa integrada

1: $j \leftarrow m$
2: **repeat**
3: $\quad j \leftarrow j - 1$
4: $\quad s \leftarrow inverse(e, r)$
5: $\quad$ **if** $s \in f$ **and** $j = 0$ **then**
6: $\quad\quad r \leftarrow next(r)$
7: $\quad\quad j \leftarrow m$
8: $\quad$ **end if**
9: **until** $s \notin f$
10: $f \leftarrow f \cup \{s\}$
11: **return** s

Capítulo 9

Búsqueda local

> I don't know what's going on! I don't know where I am! I know I'm supposed to be looking for someone, but I just can't remember! Can't remember...
>
> Dory - Finding Nemo

Finalmente, nos quedan por describir los detalles más relevantes que se tuvieron en cuenta para implementar el motor combinatorio de vac-o y las interfaces propuestas en la sección 7.4. Antes de eso, haremos un breve repaso por los diferentes paradigmas de búsqueda que nos permita justificar la decisión de utilizar búsqueda local.

9.1. Paradigmas de búsqueda

La idea fundamental de los algoritmos de búsqueda es generar y evaluar iterativamente soluciones candidatas para un problema determinado. En el caso de los "problemas de decisión", la evaluación consiste en decidir si la solución candidata es una solución del problema, mientras que para los "problemas de optimización", la evaluación consiste típicamente en determinar el valor de la función objetivo para cada solución candidata.

Generalmente, la evaluación de las soluciones candidatas depende del problema y suele ser relativamente fácil de implementar[1]. Luego, lo que diferencia fundamentalmente a los distintos algoritmos de búsqueda es la forma en que las soluciones candidatas son generadas. Los paradigmas de búsqueda más comunes son:

- Perturbativa vs. Constructiva
- Local vs. Sistemática

Aunque no es necesariamente así, suelen relacionarse la búsqueda local con la búsqueda perturbativa y la búsqueda sistemática con la búsqueda constructiva.

[1] No es el caso de los algoritmos para la predicción inversa de estructura secundaria de RNA, donde vimos que la evaluación consiste en realizar la predicción directa y esto es $\mathcal{O}(N^3)$.

9.1.1. Perturbativa vs. Constructiva

Las soluciones candidatas de un problema de optimización, están compuestas por las *componentes de la solución*. Luego, se puede modificar una solución candidata para obtener una nueva solución, cambiando una o varias de sus componentes. Esto puede ser visto como una *perturbación* sobre las soluciones candidatas y por eso se denomina a los algoritmos de búsqueda que utilizan esta forma de generar soluciones candidatas, *algoritmos de búsqueda perturbativa*.

Usualmente en estos algoritmos, la búsqueda se realiza sobre el espacio de soluciones candidatas del problema, pero eventualmente podría ser útil también incluir soluciones candidatas parciales en el espacio de búsqueda. Esto es, soluciones candidatas donde una o varias de sus componentes no ha sido determinada. En ese caso, la tarea consiste en extender las soluciones candidatas parciales hasta construir soluciones candidatas completas y por eso se denomina a los algoritmos de búsqueda para este tipo de problemas, *algoritmos de búsqueda constructiva*.

9.1.2. Local vs. Sistemática

Otra clasificación, quizás más común, entre los distintos algoritmos de búsqueda es la distinción entre *búsqueda local* y *búsqueda sistemática*. La búsqueda sistemática recorre el espacio de búsqueda de una manera sistemática garantizando que eventualmente, o bien se encuentra una solución (óptima), o bien, se tiene la certeza de que no existe ninguna solución. Esta propiedad que distingue a los algoritmos de búsqueda sistemática se denomina *completitud*.

Por otro lado, los algoritmos de búsqueda local empiezan la búsqueda en algún punto del espacio de búsqueda y sucesivamente se mueven de la solución actual a una solución "vecina" donde cada movimiento está determinado solamente por la información que brindan la solución actual y sus "vecinos". Estos algoritmos son típicamente *incompletos* en el sentido de que no hay garantía de encontrar una solución óptima ni tampoco se puede determinar con certeza que no exista una solución. Más aún, es posible que los algoritmos evalúen mas de una vez la misma solución o inclusive queden "estancados" en ciertas partes del espacio de búsqueda.

9.2. ¿Por qué usar búsqueda local?

A primera vista, los algoritmos de búsqueda local podrían parecer inferiores a los sistemáticos debido a su *incompletitud*, pero esto no es necesariamente así. Por un lado, una de las propiedades de los algoritmos sistemáticos es la de poder determinar con certeza la existencia o no de una solución. En nuestro caso, esto no es relevante ya que el mismo virus atenuado que se quiere optimizar representa una solución al problema, por lo que la existencia de una solución no está en duda. Además, los algoritmos sistemáticos y constructivos ponen especial atención al camino que se recorre para llegar a una solución y esto tampoco es de interés en un problema de optimización como el nuestro.

Por otro lado, uno de los problemas que presentan los algoritmos de búsqueda local es determinar el punto en el espacio de búsqueda por donde empezar. Una mala elección del punto inicial podría conducir a soluciones subóptimas (mínimos o máximos locales), pero esto tampoco sería un problema en nuestro

caso, ya que el virus atenuado que se quiera optimizar representa naturalmente un buen punto de inicio por haber demostrado previamente su efectividad en la práctica.

Por último, una de las propiedades deseables en la resolución de un problema de optimización es que la calidad de las soluciones esté directamente relacionada con el tiempo de ejecución del algoritmo y esto es generalmente una propiedad que se da naturalmente en los algoritmos de búsqueda local.

9.3. Implementación

Como ya mencionamos en la sección 7.4, dada la definición del problema que presentamos en la sección 4.3.1, el objetivo fue capturar la estructura general de los algoritmos de búsqueda local que suelen denominarse de "mejoramiento iterativo".

A continuación presentamos el pseudo código para este tipo de algoritmos asumiendo lo siguiente:

- *init*: representa la solución inicial provista por la extensión.
- s: representa la solución actual en cada iteración.
- $explore(s)$: genera el conjunto de soluciones vecinas de s.
- $select(s, neighbors)$: selecciona una solución entre las vecinas de s.
- $done()$: un predicado que determina cuando terminar la búsqueda.
- $notify(s')$: notifica a otros objetos la solución s'.
- $f(s)$: devuelve la evaluación de la función objetivo para s.

Procedimiento 3 Búsqueda local

```
1:  s ← init
2:  neighbors ← ∅
3:  repeat
4:     neighbors ← explore(s)
5:     s' ← select(s, neighbors)
6:     if f(s) ≤ f(s') then
7:        notify(s')
8:     end if
9:     s ← s'
10: until done()
```

Claramente, la implementación de los procedimientos *explore* y *select* serán determinantes en este algoritmo y sobre ellos profundizaremos a continuación. Por otro lado, el criterio de terminación estará relacionado con el número total de iteraciones y con el número de iteraciones que pasaron desde la mejor solución encontrada. Ambos parámetros, provistos por la extensión.

Se puede trazar una analogía entre este tipo de algoritmos y buscar el máximo de una función. El algoritmo empieza en algún punto de la función, que en

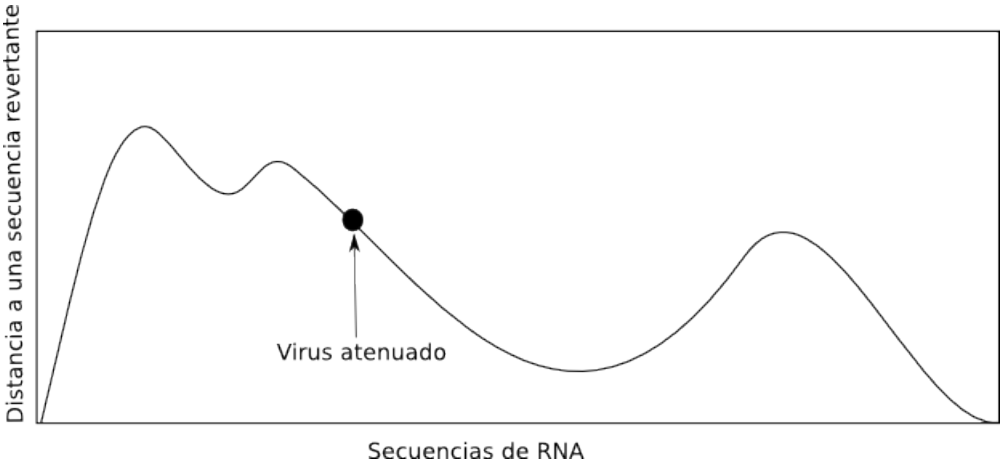

Figura 9.1: Búsqueda local por mejoramiento iterativo

nuestro caso será el virus atenuado que se pretende optimizar y luego en cada iteración, el algoritmo debe "moverse" de la solución actual a la siguiente con el objetivo de maximizar la función de evaluación.

9.3.1. El vecindario

El concepto de "vecindario" de una solución es crucial para determinar los posibles movimientos de una solución a la siguiente. Esto puede ser visto básicamente como una relación entre soluciones, y si decimos que $\mathcal{S}$ es el espacio de soluciones, entonces un vecindario será una relación $\mathcal{N} \subseteq \mathcal{S} \times \mathcal{S}$.

La interfaz propuesta en la sección 7.4 para representar esta relación entre soluciones fue **INeighborhood** y cuya principal responsabilidad es la de *explorar* el vecindario de una solución dada. Es decir, para una solución s, generar el conjunto de soluciones vecinas $\mathcal{N}(s)$.

Para implementar esta interfaz, se propuso una variante de un tipo de vecindarios que se conoce usualmente como **k-opt**, en donde dos soluciones son vecinas si difieren en a lo sumo k componentes de la solución. Recordemos que en nuestro caso, las componentes de la solución son esencialmente las regiones combinatorias representadas por la interfaz **ICombinatoryRegion** y cuya principal responsabilidad es la de generar las variantes de la región combinatoria que satisfacen las restricciones impuestas. Luego, la definición de vecindario que se propuso fue la siguiente:

Definición 5. Para $\mathcal{S}$ el espacio de soluciones, N regiones combinatorias y $m > 0$, sea $\mathcal{N} \subseteq \mathcal{S} \times \mathcal{S}$ tal que, para $s, s' \in \mathcal{S}$ el par $(s, s') \in \mathcal{N}$ si se satisfacen:

- s' difiere de s en a lo sumo una región combinatoria (**1-opt**).
- Si s' difiere de s en la región r, entonces se generaron a lo sumo m variantes de la región combinatoria r.
- Si $v_1, \ldots, v_N$ son las variantes a cada región combinatoria en la solución s', entonces $\prod_{i=1}^{N} eval(v_i) \geq umbral$.

Algunas alternativas inmediatas que surgen a esta definición son:

- Permitir que la soluciones vecinas difieran en más de una región combinatoria.
- Hacer que el número m de variantes generadas para cada región sea variable en el tiempo de forma tal de expandir el vecindario a medida que avanza la búsqueda.

9.3.2. La estrategia

Una vez que se generan las soluciones vecinas de la solución actual, se necesita algún criterio para *seleccionar* una entre todas ellas. Esto es lo que se propuso representar con la interfaz **IStrategy**. Básicamente, las implementaciones de esta interfaz determinan el criterio de selección y representan las principales diferencias entre los distintos algoritmos de búsqueda local. A continuación veremos brevemente las tres estrategias implementadas. Supongamos como antes, $\mathcal{S}$ es el espacio de soluciones, $\mathcal{N} \subseteq \mathcal{S} \times \mathcal{S}$ la relación que define el vecindario, $f : \mathcal{S} \to \mathbb{R}$ la función objetivo y $s \in \mathcal{S}$ la solución actual.

Best Improvement

Esta estrategia consiste en seleccionar una solución $s' \in \mathcal{N}(s)$ de forma tal que $f(s) \leq f(s')$ y además $f(s') = \text{máx}_{c \in \mathcal{N}(s)} f(c)$. Notar que, por un lado podría existir mas de una solución que maximice la función f para el conjunto $\mathcal{N}(s)$, en cuyo caso se seleccionará alguna de ellas. Por otro lado, esta estrategia implica generar y evaluar todas las soluciones vecinas de s, lo que podría ser eventualmente costoso dependiendo del tamaño del conjunto $\mathcal{N}(s)$.

First Improvement

A diferencia de *Best Improvement*, esta estrategia supone que los vecinos de s son generados en algún orden. Luego, en lugar de *explorar* todo el vecindario para luego seleccionar la siguiente solución, se selecciona la primer solución $s' \in \mathcal{N}(s)$ tal que $f(s) \leq f(s')$. Obviamente, esto implica un ahorro en el sentido de que no se generan y evalúan todas las soluciones vecinas de s, pero al mismo tiempo se corre el riesgo de descartar soluciones mejores a la seleccionada.

Simulated Annealing

Esta estrategia marca una diferencia importante con las anteriores como es la posibilidad de seleccionar soluciones que **no** mejoran la solución actual, pero que eventualmente podrían conducir a mejores soluciones. Esto es fundamentalmente para "escapar" de los máximos locales que podrían existir en el espacio de soluciones.

Por un lado, para seleccionar una solución peor que la actual, se debe tener en cuenta la diferencia entre sus respectivas evaluaciones, a mayor diferencia, menor debería ser la probabilidad de seleccionarla. Además, se supone que a medida que avanza la búsqueda, la calidad de las soluciones debería incrementarse, por lo tanto, la probabilidad de seleccionar una solución peor que la actual debe ser cada vez menor.

En definitiva, la estrategia consiste en seleccionar aleatoriamente una solución $s' \in \mathcal{N}(s)$ y luego decidir si s' es aceptada o no. Si $f(s) \leq f(s')$, s' se acepta automáticamente. Pero si $f(s) > f(s')$, entonces s' se acepta con probabilidad $exp(\frac{f(s')-f(s)}{T})$. Donde $T > 0$, es un parámetro que permite modular la probabilidad de aceptación a medida que avanza la búsqueda. Luego, cada algún número prefijado de iteraciones, se actualiza el valor de T haciendo $T = T \times \alpha$ para $0 < \alpha < 1$.

Parte IV

Conclusiones

Capítulo 10

Todo concluye al fin

> Pase lo que pase, dirija quien dirija, todo el mundo sabe que la camiseta 10 de la selección será mía... Para siempre.
>
> Diego Armando Maradona

10.1. Aportes

El principal aporte de este trabajo fue el análisis y la formalización del problema biológico y la posterior propuesta de cómo solucionarlo computacionalmente. En este sentido, se pudo implementar un software que sirve como una prueba de concepto aunque todavía queden funcionalidades y consideraciones a tener cuenta antes de pensar en usarlo para el diseño de vacunas atenuadas más seguras.

Por otra parte, de alguna manera esto pretende ser un aporte (muy humilde por cierto) a la "interdisciplina" como forma de aplicar los conocimientos científicos en la resolución concreta de problemas que afectan la calidad de vida de las personas.

Por último, el trabajo fue presentado en la "XXX Reunión Científica de la Sociedad Argentina de Virología (SAV)" donde recibimos muy buenas críticas y comentarios que nos incentivan a seguir trabajando.

10.2. Trabajo futuro

Como decíamos en la sección anterior, queda pendiente continuar el desarrollo del software para tener en cuenta:

- Rango de temperaturas: esto afecta la predicción de estructura secundaria de RNA y las probabilidades de mutación entre las bases de nucleótidos, por lo que es muy relevante en la probabilidad de reversión a la virulencia.
- Soporte de recombinantes: se debería contemplar la probabilidad de que el virus evolucione por combinaciones con virus homólogos y evaluar las repercusiones.

- Soporte para otros tipos de virus: tener en cuenta las particularidades de otras clases de virus de la "Clasificación de Baltimore".

Desde un punto de vista más computacional, se debería trabajar en:

- Paralelización: esto es fundamental para poder ejecutar el software con datos reales, fundamentalmente debido al costo de realizar la predicción directa e inversa de la estructura secundaria de RNA.

- Estrategias de búsqueda: se debería analizar la posibilidad de implementar otros algoritmos de búsqueda local más complejos y que probablemente conduzcan a mejores resultados.

También queda pendiente la posibilidad de desarrollar extensiones usando lenguajes de programación más "ágiles" que C++, como podría ser el caso de Python y el desarrollo de una interfaz más "amigable" para el usuario final.

Por último, se deben realizar ensayos con datos reales que permitan caracterizar de mejor manera el espacio de búsqueda y en función de eso, determinar los parámetros que mejor se ajusten al mismo.

Bibliografía

[1] Mirela Andronescu, Rosalia Aguirre-Hernandez, Anne Condon, and Holger H. Hoos. Rnasoft: a suite of rna secondary structure prediction and design software tools. *Nucleic Acids Research*, 31(13), April 2003.

[2] R. Bruce Aylward and Stephen L. Cochi. Framework for evaluating the risks of paralytic poliomyelitis after global interruption of wild poliovirus transmission. Technical report, World Health Organization, 2004.

[3] Marty R. Badgett, Alexandra Auer, Leland E. Carmichael, Colin R. Parrish, and James J. Bull. Evolutionary dynamics of viral attenuation. *Journal of Virology*, 76(20), October 2002.

[4] Anke Busch and Rolf Backofen. Info-rna - a server for fast inverse rna folding satisfying sequence constraints. *Nucleic Acids Research*, March 2007.

[5] Konstantin Chumakov and Ellie Ehrenfeld. New generation of inactivated poliovirus vaccines for universal immunization after eradication of poliomyelitis. Technical report, National Institute of Health, December 2008.

[6] J. Robert Coleman, Dimitris Papamichail, Steven Skiena, Bruce Futcher, Eckard Wimmer, and Steffen Mueller. Virus attenuation by genome-scale changes in codon pair bias. *Science*, 320:1784, June 2008.

[7] Dirección de Epidemiología. Programa Nacional de Erradicación de la Poliomielitis. Caso de poliomielitis sabin derivado en argentina. Technical report, Ministerio de Salud de la Nación, May 2009.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley, January 2005.

[9] Paul P Gardner and Robert Giegerich. A comprehensive comparison of comparative rna structure prediction approaches. *BMC Bioinformatics*, September 2004.

[10] Matthias Höchsmann. *The Tree Alignment Model: Algorithms, Implementations and Applications for the Analysis of RNA Secondary Structures*. PhD thesis, Technischen Fakultät der Universität Bielefeld, March 2005.

[11] I. L. Hofacker, W. Fontana, P. F. Stadler, L. S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of rna secondary structures. *Monatshefte für Chemie*, 125(2), 1994.

[12] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search - Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004.

[13] Nidia H De Jesus. Epidemics to eradication: the modern history of poliomyelitis. *Virology Journal*, July 2007.

[14] Adam S Lauring, Jeremy O Jones, and Raul Andino. Rationalizing the development of live attenuated virus vaccines. *Nature Biotechnology*, 28(6), June 2010.

[15] Rune B. Lyngsø and Christian N. S. Pedersen. Rna pseudoknot prediction in energy based models. *Journal of computational biology*, 2000.

[16] Robert C. Martin. Design principles and design patterns. www.objectmentor.com, 2000.

[17] Steve McConnell. *Rapid Development: taming wild software schedules*. Microsoft Press, 1996.

[18] Philip D. Minor. The molecular biology of poliovaccines. *Journal of General Virology*, 1992.

[19] Steffen Mueller, J Robert Coleman, Dimitris Papamichail, Charles B Ward, Anjaruwee Nimnual, Bruce Futcher, Steven Skiena, and Eckard Wimmer. Live attenuated influenza virus vaccines by computer-aided rational design. *Nature Biotechnology*, 28(7), July 2010.

[20] Marco Vignuzzi, Emily Wendt, and Raul Andino. Engineering attenuated virus vaccines by controlling replication fidelity. *Nature Medicine*, 14(2):154, February 2008.

[21] Rebecca Wirfs-Brock and Alan McKean. *Object Design: Roles, Responsibilities and Collaborations*. Addison-Wesley, 2003.

[22] Michael Zuker and David Sankoff. Rna secondary structures and their prediction. *Bulletin of Mathematical Biology*, 46(4), 1984.

[23] Michael Zuker and Patrick Stiegler. Optimal computer folding of large rna sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9(1), 1981.

Apéndice A

Acrónimos

RNA	Ácido ribonucleico . . . 1
DNA	Ácido desoxirribonucleico . . . 7
OPV	*Oral Polio Vaccine* . . . 7
WHO	*World Health Organization* . . . 7
cVDPV	poliovirus circulantes derivados de la vacuna . . . 8
VAPP	parálisis poliomielítica asociada a la vacuna oral . . . 8
FuDePAN	Fundación para el Desarrollo de la Programación en Ácidos Nucleicos . . . 8
vac-o	*Combinatory Vaccine Optimizer* . . . 8
OOP	*Object Oriented Programming* . . . 9
GPLv3	*GNU General Public License v3* . . . 9
SVN	Subversion . . . 9
A	Adenina . . . 12
G	Guanina . . . 12
T	Timina . . . 12
C	Citosina . . . 12
U	Uracilo . . . 12
mRNA	*messenger RNA* . . . 14
rRNA	*ribosomal RNA* . . . 14
tRNA	*transfer RNA* . . . 14
PV1	Poliovirus tipo 1 . . . 16
PV2	Poliovirus tipo 2 . . . 16
PV3	Poliovirus tipo 3 . . . 16
mfe	*minimal free energy* . . . 18
CSP	*Constraint Satisfaction Problem* . . . 19
ORF	*Open Reading Frame* . . . 13
UTR	*Untranslated Region* . . . 13

IRES *Internal Ribosomal Entry Site* 13
RDD *Responsibility-Driven Design* 34
SRP *Single Responsability Principle* 30
OCP *Open-Close Principle* 30
LSP *Liskov Substitution Principle* 30
ISP *Interface Segregation Principle* 31
DIP *Dependency Inversion Principle* 31
(+)ssRNA virus *positive-sense single-stranded RNA virus* 15
SAV Sociedad Argentina de Virología 52

Anexos

Especificación de Requerimientos de Software

Santiago Videla

24 de septiembre de 2010

Índice

1. Introducción

1.1. Propósito

El propósito de este documento es la especificación de requerimientos de software en el marco de la tesis de grado de la carrera Lic. en Cs. de la Computación de la FaMAF - UNC, **"Diseño de vacunas atenuadas con menor probabilidad de sufrir reversión a la virulencia"**. Los requerimientos del software son provistos por integrantes de FuDePAN en su carácter de autores intelectuales de la solución que se pretende implementar y colaboradores de dicha tesis.

A continuación se enumeran las personas involucradas en el desarrollo de la tesis y que por lo tanto, representan la principal audiencia del presente documento.

- Dra. Laura Alonso Alemany: Directora de tesis, FaMAF
- Daniel Gutson: Colaborador de tesis, FuDePAN
- Santiago Videla: Tesista, FaMAF

1.2. Alcance

El producto que se especifica en este documento se denomina **"vac-o"** y su principal objetivo es encontrar secuencias genómicas de patógenos que mantengan funcionalidad como vacunas con una probabilidad mínima de sufrir reversión a la virulencia. El producto final debe proveer al usuario, la capacidad de hacer uso del software mediante extensiones o *plugins* que implementen las características particulares para cada patógeno.

La principal responsabilidad de vac-o es la de proveer a las extensiones, un motor combinatorio de secuencias genómicas utilizando diferentes estrategias computacionales que conduzcan a la optimización de los cálculos. Por otro lado, las extensiones son responsables de evaluar las secuencias generadas asignando un puntaje a cada una con el fin de que vac-o genere el listado de secuencias resultante.

En su versión inicial, vac-o incluye una extensión para la vacuna Sabin contra la poliomielitis que podría ser tomada como guía para futuras extensiones.

1.3. Descripción general del documento

La estructura de este documento sigue las recomendaciones de la "Guía para la especificación de requerimientos de la IEEE" (IEEE Std 830-1998).

En la sección 2 se presenta una descripción general de vac-o, sus principales funcionalidades, interfaces y perfiles de usuarios.

En la sección 3 se detallan los requerimientos funcionales específicos de vac-o y los principales atributos que debe cumplir el software.

2. Descripción General

2.1. Perspectiva del Producto

Al momento de la confección de este documento, no existen productos de software que brinden las funcionalidades que se pretenden implementar. En este sentido, vac-o representa una innovación en el área de la bioinformática, cuyo principal objetivo es la optimización de vacunas basadas en virus atenuados.

Uno de los problemas de estas vacunas es su potencialidad de revertir a la virulencia. En sucesivas replicaciones, los virus atenuados pueden acumular mutaciones que les devuelvan el carácter patogénico y de esa manera podrían producir enfermedad en el vacunado o sus contactos.

Para lograr su objetivo, vac-o debe ser capaz de encontrar las variantes del virus con menor probabilidad de revertir a la patogenia. Partiendo de la secuencia genómica que se encuentra en la cepa vacunal, se deben buscar las modificaciones de esta secuencia que cumplan las siguientes propiedades:

- Que originen una respuesta inmune protectora contra el virus salvaje.
- Que tengan muy baja probabilidad de mutar hasta convertirse en un virus patógeno.

Entre todas las secuencias encontradas, vac-o debe construir una lista o *ranking* de secuencias con sus respectivos puntajes. Para lograr esto, se deberá implementar una extensión que brinde la información y las características propias del virus. En particular, cada extensión debe especificar las regiones combinatorias de interés y ser capaz de asignar un puntaje a cada secuencia comparándola con la secuencia original.

En la Figura 1 se presenta el flujo de trabajo y funcionamiento general de vac-o. Notar que este esquema es puramente conceptual y no hace ninguna referencia a los detalles de implementación. La intención es clarificar al lector los diferentes "actores" y sus responsabilidades.

2.1.1. Interfaces del Sistema

No se registran requerimientos.

2.1.2. Interfaces de Usuario

La interfaz con el usuario final consiste de dos formularios y un listado con las secuencias resultantes. Inicialmente, el usuario de vac-o deberá indicar la ubicación o *path* de la extensión que desea usar. El siguiente paso, será completar un formulario con los datos requeridos por la extensión elegida. Finalmente, se da comienzo a la ejecución y se recibe como resultado, la lista de secuencias con sus respectivos puntajes.

2.1.3. Interfaces de Hardware

No se registran requerimientos.

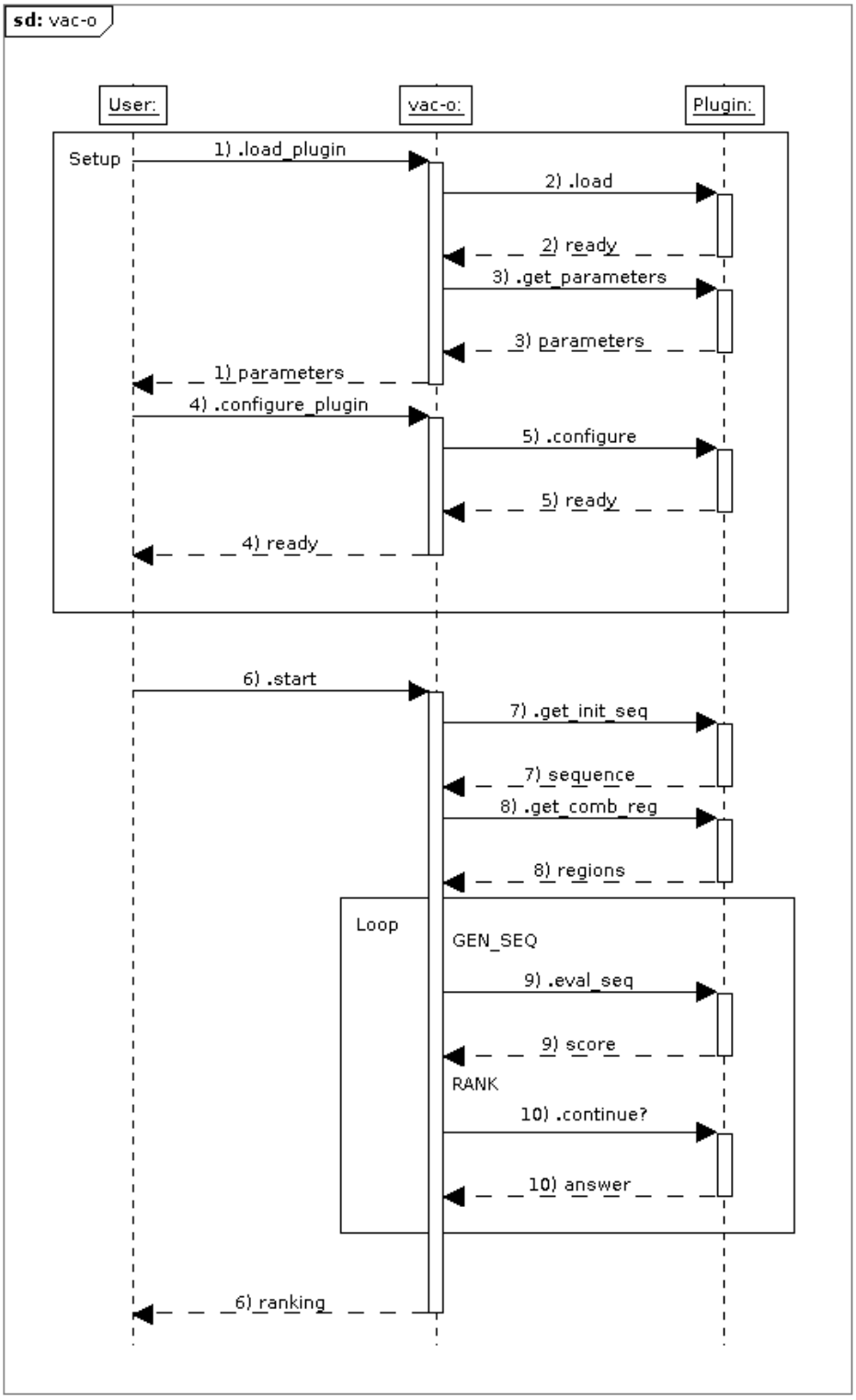

Figura 1: Flujo de trabajo de vac-o

2.1.4. Interfaces de Software

- BioPP: Se debe utilizar la librería BioPP para realizar el *folding* de secuencias genómicas, predicción de la estructura secundaria y cálculo de distancias.
- Boost.Python: Se debe utilizar Boost.Python para brindar a los desarrolladores de extensiones, la posibilidad de hacerlo utilizando el lenguaje de programación Python.

2.1.5. Interfaces de Comunicaciones

No se registran requerimientos.

2.1.6. Restricciones de Memoria

No se registran restricciones de memoria para la ejecución del software. No obstante, se debe tener en cuenta que dada la naturaleza y la complejidad del problema, el tiempo de cálculo estará directamente relacionado con la memoria disponible.

2.1.7. Operaciones

No se registran requerimientos.

2.1.8. Requerimientos de Instalación

No se registran requerimientos.

2.2. Funciones del Producto

2.2.1. Configuración inicial

Inicialmente, vac-o presenta al usuario final una pantalla con un formulario donde el usuario debe indicar la ubicación de la extensión que desea usar. A continuación, vac-o carga la extensión en memoria y presenta al usuario otro formulario con los campos requeridos por la extensión cargada. Una vez que los datos son validados, vac-o configura la extensión con los valores enviados por el usuario y queda listo para comenzar la ejecución cuando el usuario lo indique.

2.2.2. Configuración de vac-o

Antes de dar comienzo a la generación de secuencias genómicas, vac-o debe solicitar a la extensión cargada los siguientes valores:

- Secuencia inicial: La secuencia genómica que se encuentra en la cepa vacunal y se desea mejorar.
- Regiones combinatorias: Las regiones de la secuencia inicial que resultan de interés, indicando sus posiciones de inicio y fin, y qué tipo de regiones son cada una (restricciones en base a código genético, o en base a estructura secundaria).

- Estrategias de búsqueda: La estrategia que se debe utilizar para la generación o búsqueda de nuevas secuencias.

2.2.3. Generación de secuencias

Luego de las configuraciones básicas, vac-o esta en condiciones de generar nuevas secuencias genómicas que cumplan las restricciones que hayan sido impuestas. Para esto, se deben calcular las posibles "variantes" de cada región combinatoria (teniendo en cuenta el tipo de región y sus posibles intersecciones) y de acuerdo a la estrategia de búsqueda, construir nuevas secuencias que serán evaluadas posteriormente por la extensión. Dado que el espacio de búsqueda podría ser eventualmente muy grande, el criterio de parada para la generación de secuencias también debe ser provisto por la extensión.

Notar que esta funcionalidad es el "corazón" de vac-o y es donde se encuentra la mayor complejidad del problema a resolver.

Supongamos $N \geqslant 1$ y $R_1, \cdots, R_N$ regiones combinatorias. Luego, tendremos:

$$\begin{array}{c} M_{1\cdot 1}, ..., M_{1\cdot j_1} \text{ mutaciones de } R_1 \\ M_{2\cdot 1}, ..., M_{2\cdot j_2} \text{ mutaciones de } R_2 \\ \cdots \\ M_{N\cdot 1}, ..., M_{N\cdot j_N} \text{ mutaciones de } R_N \end{array}$$

con $j_1, \cdots, j_N \geqslant 1$. Es decir, que el número de posibles secuencias que vac-o "debería" evaluar sera igual al producto $j_1 \times j_2 \times \cdots \times j_N$.

Haciendo uso de diferentes estrategias de búsqueda, combinadas con las funciones de evaluación provistas por la extensión, vac-o intentara optimizar esta generación de secuencias.

2.2.4. Evaluación de secuencias

De acuerdo a los puntajes asignados por la extensión a cada una de las secuencias generadas, vac-o debe construir un listado o *ranking* de secuencias que será dado al usuario como resultado final de la ejecución.

2.3. Características de Usuarios

Se identifican 3 tipos de usuarios de vac-o:

- Final: que solo interactúa a través de la interfaz gráfica.
- Extensionista: que posee los conocimientos de programación suficientes como para implementar extensiones de vac-o. A los fines de ampliar los potenciales usuarios dentro de esta categoría, es que se ofrece la posibilidad de desarrollar extensiones usando el lenguaje Python.
- Contribuidor: que contribuye al código fuente de vac-o realizando mejoras o desarrollando nuevas funcionalidades.

2.4. Restricciones

El producto debe ser desarrollado utilizando el lenguaje de programación C++ y bajo la licencia de software GPLv3.

2.5. Suposiciones y Dependencias

- Sistema operativo: GNU/Linux

2.6. Trabajo Futuro

Probablemente una de las principales mejoras a la primer versión de vac-o, será realizar las modificaciones necesarias para permitir la ejecución del software en paralelo y de esta manera reducir los tiempos de cálculo. Por otro lado, se deberá profundizar en el desarrollo de extensiones para diferentes tipos de vacunas y virus.

3. Requerimientos

3.1. Funciones del Sistema

3.1.1. Configuración inicial

El objetivo de esta función es la carga en memoria de la extensión que se desea utilizar para ejecutar vac-o.

1. **Carga de la extensión:** El sistema recibe la ubicación del archivo *.so* y lo carga en memoria. Si la carga es exitosa, se devuelven los parámetros requeridos por la extensión (*[(nombre, tipo, validaciones, defecto)]*) para su posterior configuración. En caso de que la carga de la extensión fracase, se devuelve un mensaje de error.

2. **Validación de valores:** La interfaz de usuario es responsable de la validación de los datos ingresados por el usuario para la configuración de la extensión. Para esto, se deberán usar los diferentes *criterios de validación* para cada parámetro y solo cuando la validación sea exitosa, los valores son enviados al sistema. Caso contrario, se debe presentar un mensaje de error al usuario.

3. **Configuración de la extensión:** El sistema recibe de la interfaz de usuario, la lista de parámetros requeridos por la extensión (*[(nombre, valor)]*) y se asignan los valores en la extensión. Finalmente, se devuelve un mensaje de éxito indicando que vac-o está listo para comenzar la ejecución.

4. **Tamaño del ranking:** El sistema recibe de la interfaz de usuario, el tamaño del ranking de secuencias que dará como resultado de la ejecución.

3.1.2. Configuración de vac-o

El objetivo de esta función es solicitar a la extensión cargada en memoria, la información necesaria para comenzar la ejecución del motor combinatorio.

1. **Secuencia inicial:** El sistema solicita a la extensión cargada en memoria, la secuencia de ARN que se desea optimizar. La extensión debe devolver la secuencia y el sistema la almacena en memoria.

2. **Regiones combinatorias:** El sistema solicita a la extensión cargada en memoria, las regiones combinatorias de interés para la optimización. La extensión debe devolver las regiones (*[(inicio, fin, tipo, evaluador)]*) y el sistema inicializa cada región en memoria. Para cada región, el parámetro *tipo* debe ser uno de los siguientes:

   - Estructura secundaria (SS): Las posibles variantes de la región deben conservar la estructura secundaria.
   - Código genético (GC): Las posibles variantes de la región deben resultar silentes en términos de expresión aminocídica.
   - Personalizada (CU): La extensión debe proporcionar la implementación de la región combinatoria.

   Además, se debe considerar la posibilidad de que en el futuro se necesiten nuevos tipos de regiones combinatorias.

   El parámetro *evaluador* debe ser una función $f : Secuencia \to (0, 1)$ que será utilizada por el sistema para determinar localmente la "bondad" de las diferentes variantes de cada región.

3. **Estrategia de búsqueda:** El sistema solicita a la extensión un *umbral* de "bondad" y debe implementar diferentes estrategias de optimización, asegurando que se cumpla lo siguiente:

   Sea n el numero de regiones combinatorias, s_i la secuencia seleccionada de la región i y $f_i : Secuencia \to (0, 1)$ la función de evaluación de la región i, con $i = 1, ..., n$.

   $$\prod_{i=1}^{n} f_i(s_i) \geq umbral$$

4. **Regiones de validación:** El sistema solicita a la extensión cargada en memoria, las regiones sobre las que se debe hacer el "control de calidad". La extensión debe devolver las regiones (*[(inicio, fin, prueba, criterio)]*) y el sistema inicializa cada región en memoria. Para cada región, el parámetro *prueba*, debe ser uno de los siguientes:

   - Mutaciones sistemáticas (ALL): Se calculan todas las posibles mutaciones.
   - Mutaciones al azar (RAND): Se calculan N mutaciones al azar.
   - Mutaciones personalizadas (CU): La extensión debe proporcionar la implementación.

   El parámetro *criterio* debe ser uno de los siguientes:

   - Similitud estructural (SS): La estructura secundaria de las mutaciones debe tener un porcentaje de similitud mayor o igual que un valor dado.
   - Disimilitud estructural (DS): La estructura secundaria de las mutaciones debe tener un porcentaje de similitud menor o igual que un valor dado.

   Además la extensión debe especificar la profundidad M con la que se desea realizar el "control de calidad".

3.1.3. Generación de secuencias

El objetivo de esta función es la construcción de nuevas secuencias genómicas que cumplan con las restricciones impuestas.

1. **Generar variantes de región combinatoria:** Para cada región combinatoria, el sistema debe ser capaz de generar las posibles variantes que cumplan con las restricciones impuestas por el tipo de región y considerando las posibles intersecciones con las demás regiones. Además, debe ser posible el uso de diferentes librerías externas a ser determinadas por la extensión.

2. **Generar secuencias:** A partir de las variantes de cada región combinatoria, se deben construir nuevas secuencia genómicas, reemplazando en la secuencia inicial, cada región por su variante.

   Esta funcionalidad representa el corazón del sistema y consiste en la construcción de un árbol de (sub)secuencias derivadas de cada una de las regiones combinatorias, donde cada hoja del árbol determinará una nueva secuencia genómica completa. Luego, el sistema debe ser capaz de generar y recorrer el árbol basándose en la información brindada por la extensión.

   El criterio de terminación en la generación de secuencias (recorrido del árbol) debe ser provisto por la extensión.

3. **Validar secuencias:** Cada secuencia generada por el sistema debe ser sometida al "control de calidad" y solo aquellas que lo pasen exitosamente serán consideradas para una posterior evaluación.

   El procedimiento consiste en generar y recorrer un árbol de mutaciones acumuladas por cada "región de validación", que tendrá a lo sumo, profundidad M (dado por la extensión).

   Diremos que una región pasa el "control de calidad" cuando sea posible generar el árbol completo. Es decir, cuando todos los nodos (mutantes) del árbol, pasen el criterio de prueba especificado. Si alguna región no pasa el "control de calidad", entonces la secuencia se descarta.

   Tanto el procedimiento para generar las mutaciones en cada nivel del árbol, como el criterio de prueba al que se somete cada mutación, están determinados por la "región de validación".

3.1.4. Evaluación de secuencias

El objetivo de esta función es la evaluación y posterior construcción de un *ranking* de secuencias genómicas.

1. **Asignar puntaje a una secuencia:** El sistema solicita a la extensión cargada en memoria, un puntaje para una secuencia dada. La extensión debe evaluar la secuencia recibida comparándola con la secuencia inicial y devolver un puntaje.

2. **Construir *ranking* de secuencias:** El sistema debe construir un *ranking* de secuencias basándose en los puntajes asignados por la extensión. Este listado de secuencias es además, la salida final de la ejecución.

3.2. Restricciones de Rendimiento

No se registran requerimientos.

3.3. Base de Datos

No se registran requerimientos.

3.4. Restricciones de Diseño

3.4.1. Cumplimiento de Estándares

El producto debe cumplir con los siguientes principios de diseño de la programación orientada a objetos. Los primeros 5, son también conocidos por el acrónimo "**SOLID**".

- **S**ingle responsibility principle (SRP)
- **O**pen/closed principle (OCP)
- **L**iskov substitution principle (LSP)
- **I**nterface segregation principle (ISP)
- **D**ependency inversion principle (DIP)
- Law of Demeter (LoD)

Además el producto debe cumplir con el estandar ANSI C++ y el "coding style" provisto por FuDePAN.

3.5. Atributos del Software

Se requiere que el código del producto tenga un 80 % de cobertura con pruebas automatizadas.

Apendices

A. Definiciones, Acrónimos y Abreviaturas

- **vac-o**: Combinatory Vaccine Optimizer.
- **FaMAF**: Facultad de Matemática, Astronomía y Física.
- **UNC**: Universidad Nacional de Córdoba.
- **FuDePAN**: Fundación para el Desarrollo de la Programación en Ácidos Nucleicos.
- **API**: Application Programming Interface.
- **GPL**: General Public License.
- **IEEE**: Institute of Electrical and Electronics Engineers
- **SOLID**: acrónimo nemotécnico introducido por Robert C. Martin en la década de 2000, que representa cinco principios básicos de programación y diseño orientado a objetos.
- **LoD**: Law of Demeter, principio de diseño orientado a objetos para lograr "bajo acoplamiento".

B. Referencias

- C++: Lenguaje de programación.
  `http://www.cplusplus.com`
- Python: Lenguaje de programación interpretado.
  `http://www.python.org`
- BioPP: Librería C++ para biología molecular
  `http://code.google.com/p/biopp`
- Boost.Python: Librería C++ para la interacción con Python.
  `http://www.boost.org/doc/libs/1_42_0/libs/python/doc/index.html`
- QT: Librería C++ para el desarrollo de interfaces gráficas.
  `http://qt.nokia.com/products`
- GPL: General Public License.
  `http://www.gnu.org/licenses/gpl.html`
- IEEE STD 830-1998: Guía para la especifiación de requerimientos.
  `http://standards.ieee.org/reading/ieee/std_public/description/se/830-1998_desc.html`
- SOLID: "Design Principles and Design Patterns", Robert C. Martin.
  `http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf`

Especificación de Diseño de Software

Santiago Videla

12 de octubre de 2010

Índice

1. Introducción

1.1. Propósito

El propósito de este documento es la especificación de diseño de software para la primer versión del producto "vac-o".

La confección de este documento se enmarca dentro del desarrollo de la tesis de grado de la carrera Lic. en Cs. de la Computación de la FaMAF - UNC, **"Diseño de vacunas atenuadas con menor probabilidad de sufrir reversión a la virulencia"** a cargo de Santiago Videla, con la dirección de la Dra. Laura Alonso i Alemany (FaMAF) y la colaboración de Daniel Gutson (FuDePAN).

El documento esta dirigido a las personas involucradas en el desarrollo de la tesis como así también a los colaboradores de FuDePAN que eventualmente podrían participar en las etapas de desarrollo y mantenimiento del software.

1.2. Descripción general del documento

En la sección 2 se mencionan los objetivos, la metodología adoptada y las dependencias del diseño.

En la sección 3 se muestra la arquitectura del sistema con sus principales componentes e interacciones.

En la sección 4 se presenta el diseño de alto nivel del sistema, sus interfaces y paquetes principales.

En la sección 5 se presenta el diseño de bajo nivel del sistema, las clases concretas y sus relaciones para cada paquete.

2. Consideraciones de Diseño

2.1. Objetivos

Principalmente se pretende lograr un diseño que cumpla con los principios fundamentales del diseño orientado a objetos, comúnmente conocidos por el acrónimo "SOLID" [1].

En particular, se pone especial énfasis en respetar los principios SRP (*Single Responsability Principle*), OCP (*Open-Closed Principle*) y DIP (*Dependency Inversion Principle*) debido a su importancia para obtener un sistema que sea fácilmente extensible y configurable con el fin de satisfacer las necesidades de los usuarios.

2.2. Metodología

La metodología utilizada para realizar el análisis y descripción del diseño se denomina "Diseño dirigido por responsabilidades" [2]. Esta técnica se enfoca en *qué* acciones (responsabilidades) deben ser cubiertas por el sistema y que objetos serán los responsables de llevarlas a cabo. *Cómo* se realizara cada acción, queda en un segunda plano.

2.3. Dependencias

Se asume para la confección de este diseño, el acceso a diferentes librerías externas que serán fundamentales para el correcto funcionamiento del sistema en su conjunto. Respetando la metodología adoptada, no se hará referencia directa a una u otra librería, sino a los servicios que las mismas debe ser capaces de proveer al sistema.

2.4. Herramientas y convenciones

Se utiliza UML[3] como lenguaje de modelado y ArgoUML[4] como herramienta para la confección de diagramas. Además se adopta la convención de nombrar a las interfaces anteponiendo una *"I"* al nombre de la clase concreta que la implementa (interface: *"IPersona"*, clase concreta: *"Persona"*).

3. Arquitectura del Sistema

En la Figura 1 se presenta un diagrama de la arquitectura del sistema y la interacción entre sus componentes principales. A continuación se da una breve descripción de cada uno de ellos:

- **Main:** Representa el componente principal en términos de ejecución del sistema. Comprende principalmente, la inicialización y configuración de otros componentes.
- **Plugin:** Representa la implementación de las características propias de la vacuna que se desea optimizar. Brinda información de configuración inicial como así también, los criterios para evaluar una secuencia.
- **CombinatoryEngine:** Representa el motor combinatorio del sistema encargado de encontrar, nuevas secuencias que sean candidatas a optimizar la atenuación de la vacuna.
- **QAEngine:** Representa el motor de control de calidad del sistema encargado de decidir, si una secuencia candidata pasa o no dicho control.
- **Ranker:** Representa el componente encargado de mantener un "ranking" de secuencias.
- **libRNA:** Representa el componente que provee al sistema funcionalidades para la manipulación de secuencias de ARN ("folding" directo e inverso) utilizando librerías externas (Vienna Package, UNAFold, entre otras)

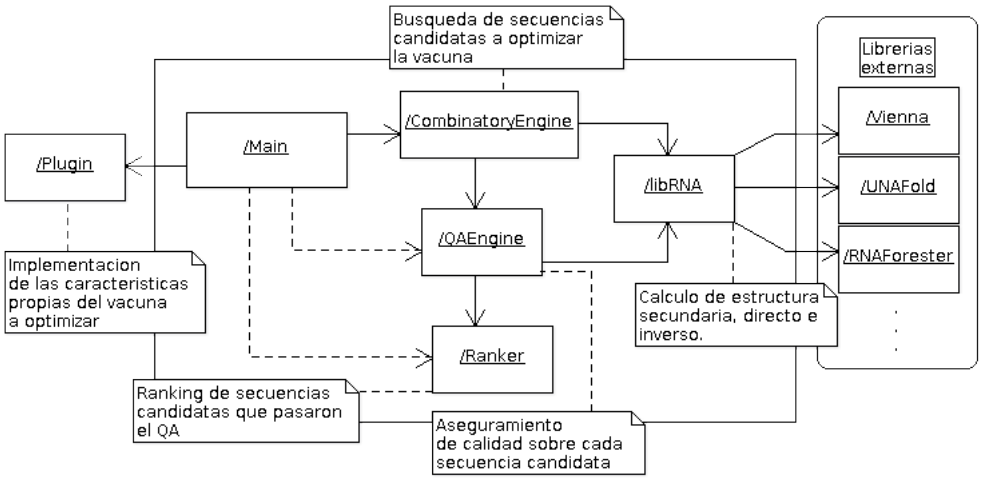

Figura 1: UML - Arquitectura

4. Diseño de alto nivel

4.1. Interfaces - Responsabilidades - Colaboradores

En esta sección se presentan las principales interfaces que intervienen en el sistema, sus respectivas responsabilidades y colaboradores. En la Figura 2 se puede ver el diagrama de clases correspondiente.

Finalmente, en la Figura 3 se presenta el diagrama de secuencia correspondiente a la comunicación entre las principales entidades del sistema.

4.1.1. IPluginAdmin

Responsabilidad: Administrar las extensiones del sistema (archivos *.so*).

1. Cargar extensión.

4.1.2. IPlugin

Responsabilidad: Brindar la información y servicios particulares para una vacuna determinada.

1. Proveer la lista de parámetros requeridos por la extensión.
2. Proveer la solución inicial conteniendo la secuencia de ARN que se encuentra en la cepa vacunal.
3. Proveer las regiones combinatorias que se deben usar para buscar mejoras a la vacuna.
4. Proveer el umbral que se debe usar para determinar la bondad de las secuencias obtenidas de las regiónes combinatorias.
5. Proveer las regiones de validación que se deben usar para realizar el control de calidad.
6. Proveer el vecindario para la búsqueda local.
7. Proveer la estrategia de búsqueda local.
8. Determinar si se continua buscando secuencias o no.
9. Evaluar las soluciones candidatas.
10. Descargar la extensión.

4.1.3. IFold

Responsabilidad: Proveer al sistema el “folding” directo de secuencias ARN

Colaboradores:

1. Vienna Package, o UNAFold, u otros.

4.1.4. IFoldInverse

Responsabilidad: Proveer al sistema el “folding” inverso de secuencias ARN

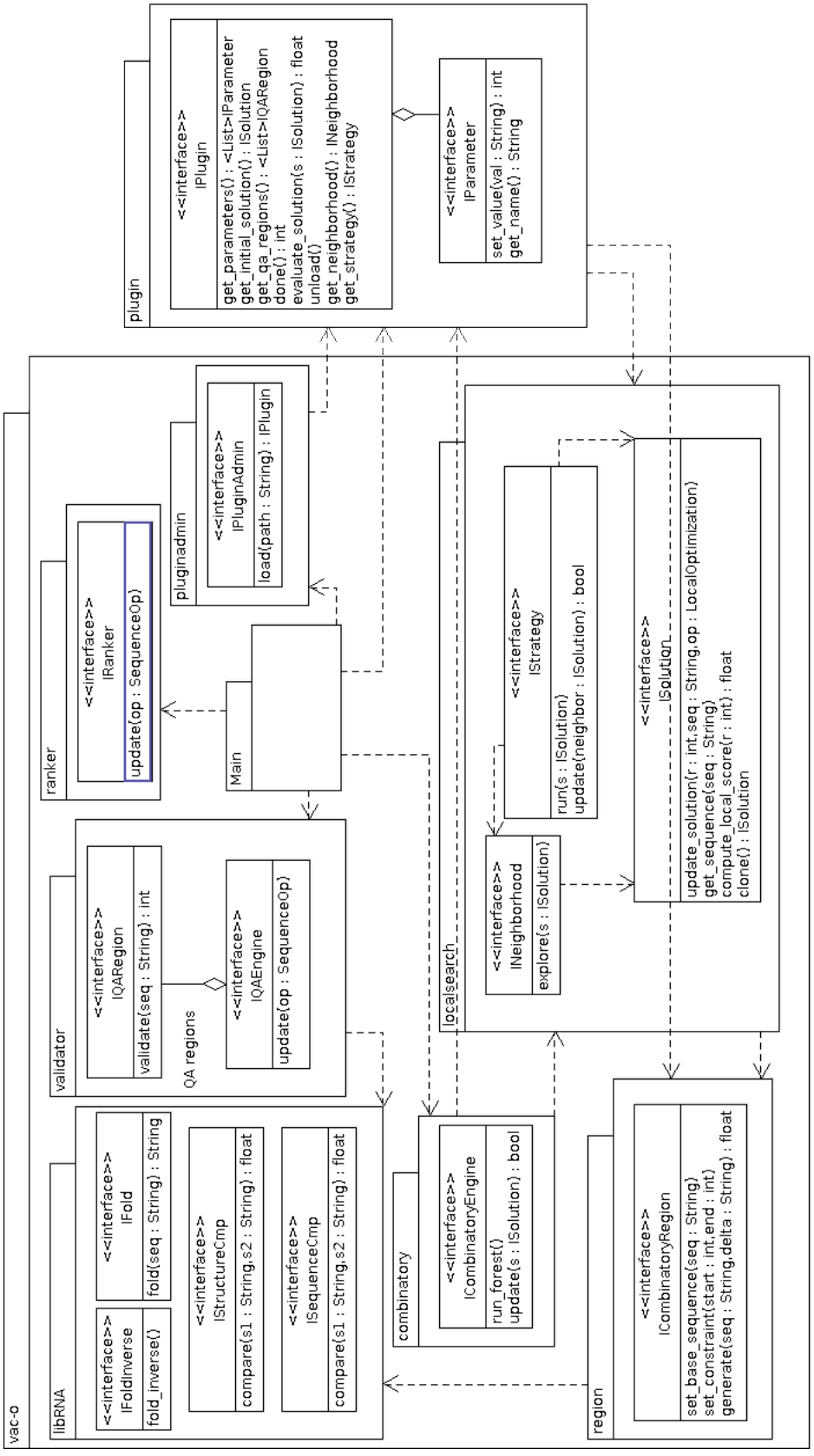

Figura 2: UML - Interfaces

Colaboradores:

1. Vienna Package u otros.

4.1.5. IStructureCmp

Responsabilidad: Proveer al sistema la comparación de estructuras secundarias.

Colaboradores:

1. RNAForester u otros.

4.1.6. ISequenceCmp

Responsabilidad: Proveer al sistema la comparación de secuencias de ARN.

Colaboradores:

1. Vienna Package u otros.

4.1.7. ICombinatoryRegion

Responsabilidad: Calcular las secuencias que mantengan determinadas propiedades de una secuencia original.

1. Devolver la siguiente secuencia, el cambio realizado y la evaluación local del cambio.

Colaboradores:

1. IFold, IFoldInverse, IStructureCmp, ISequenceCmp

4.1.8. ISolution

Responsabilidad: Representar una solución candidata en el espacio de busqueda.

1. Proveer la secuencia completa de la solución.
2. Actualizar una componente (región) de la solución y la secuencia completa resultante.
3. Calcular la evaluación local de la solución como producto de la evaluación de sus componentes (regiones).

4.1.9. INeighborhood

Responsabilidad: Explorar el vecindario de una solución.

Colaboradores:

1. **ICombinatoryRegion:** Consulta la siguiente secuencia de cada región combinatoria.

4.1.10. IStrategy

Responsabilidad: Establecer la política para pasar de una solución a la siguiente y notificar cada solución que mejora a las anteriores.

1. Seleccionar una solución entre los vecinos de la solución actual.

Colaboradores

1. **INeighborhood:** Explora el vecindario para cada solución.

4.1.11. ICombinatoryEngine

Responsabilidad: Generar secuencias candidatas a partir de las soluciones encontradas por la búsqueda local.

1. Iniciar la búsqueda local.
2. Notificar nuevas secuencias candidatas.

Colaboradores:

1. **IStrategy:** Ejecuta la búsqueda local.

4.1.12. IQARegion

Responsabilidad: Realizar el control de calidad para una región de validación.

1. Calcular y validar mutaciones acumuladas de la región hasta alcanzar la profundidad deseada.

Colaboradores:

1. IFold

4.1.13. IQAEngine

Responsabilidad: Realizar el control de calidad para una secuencia candidata.

1. Determinar si una secuencia candidata aprueba o no el control de calidad para todas sus regiones de validación.

Colaboradores:

1. **IQARegion:** Consulta si la región de validación aprueba o no el control de calidad.

4.1.14. IRanker

Responsabilidad: Mantener un *ranking* de secuencias en base a evaluación global de cada una de ellas.

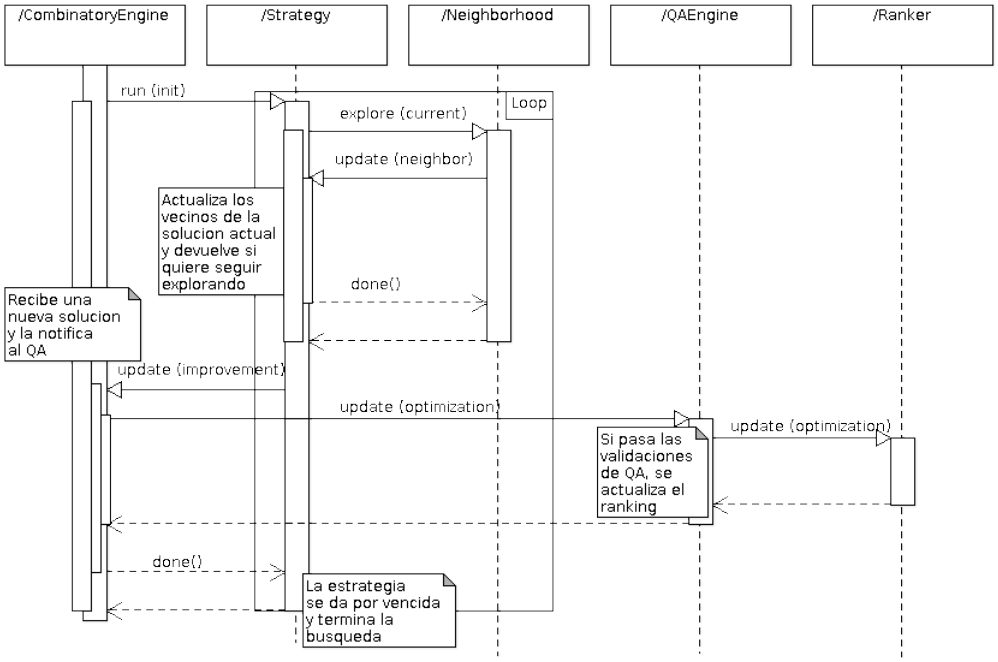

Figura 3: UML - Pasaje de mensajes

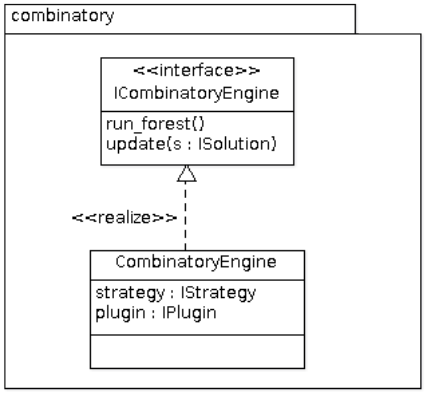

Figura 4: UML - Combinatory

5. Diseño de bajo nivel

5.1. Paquetes y clases concretas

En esta sección se presentan las clases concretas que implementan las interfaces presentadas en la sección 4. Para mayor claridad, se dividieron los diagramas UML por paquetes.

5.1.1. Combinatory

En la Figura 4 se puede ver el diagrama de clases para el paquete *combinatory*. Este paquete forma parte del componente "CombinatoryEngine" en la Figura 1 de la sección 3.

La responsabilidad de este paquete es brindar al sistema, la interfaz al motor combinatorio ocultando la implementación de la estrategia de búsqueda utilizada.

5.1.2. LocalSearch

En la Figura 5 se puede ver el diagrama de clases para el paquete *localsearch*. Este paquete forma parte del componente "CombinatoryEngine" en la Figura 1 de la sección 3.

La responsabilidad de este paquete es brindar al motor combinatorio, la implementación de diferentes estrategias de búsqueda local para la optimización de una solución inicial (la vacuna a optimizar). Inicialmente, se consideran la "familia" de algoritmos de búsqueda local por "mejoramiento iterativo". Algunos de estos algoritmos son:

- HillClimbing (FirstImprovement o BestImprovement)
- Simulated Annealing

Cada una de estas estrategias difieren una de las otras, en como se produce el paso de una solución a la siguiente. La interface "IStrategy" define los métodos que deben implementar las clases para cada estrategia.

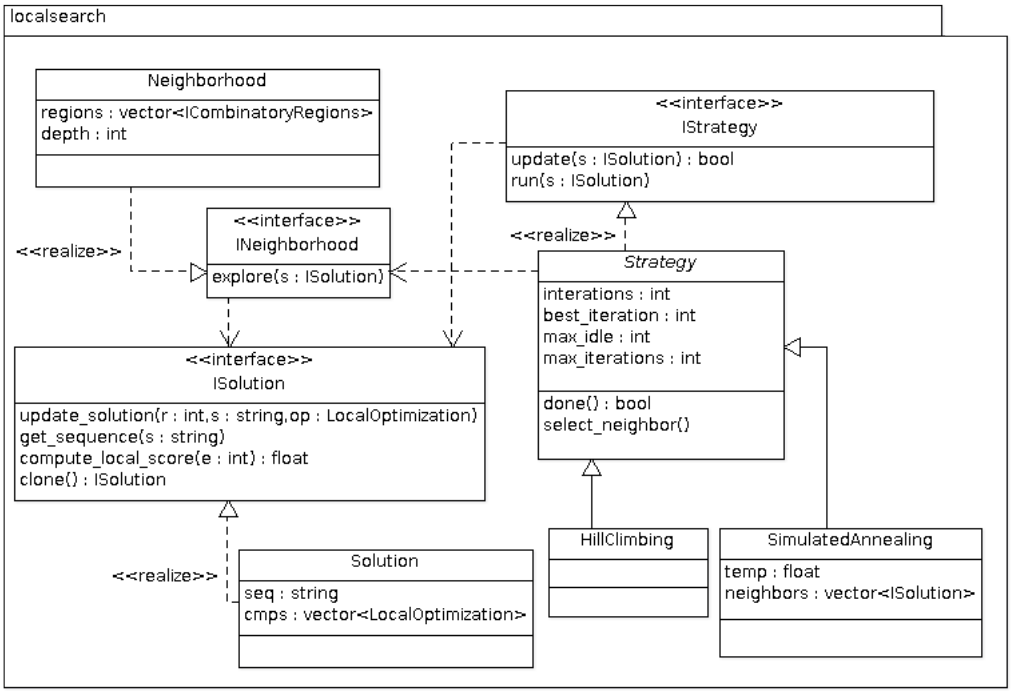

Figura 5: UML - LocalSearch

Otro elemento fundamental para la búsqueda local, es la definición de lo que se denomina el vecindario de cada solución. Es decir, una relación entre soluciones del problema, que permita ir de una solución a la siguiente. Esto esta representado por la interface "INeighborhood". Cada clase concreta que la implemente, define un tipo diferente de vecindario. Un ejemplo concreto de vecindario puede ser el siguiente:

Sea S el conjunto de todas las soluciones (secuencias de ARN que satisfacen las restricciones impuestas), definimos para $0 < i$, $N(i) \subseteq S \times S$ tal que, el par $(s, s') \in S \times S$, pertenece a $N(i)$ si ocurre lo siguiente:

1. s' difiere de s en a los sumo una región
2. Si s' difiere de s en la región k, entonces se generaron a lo sumo i variantes para llegar de s_k a s'_k
3. Sean k_i las regiones de s', entonces se cumple $\prod_i score(k_i) \geq cutoff$

Donde $score$ es la función que devuelve la evaluación local de una región y $cutoff \in (0, 1)$ es el umbral de aceptación.

Las diferentes combinaciones entre definiciones de vecindarios y estrategias impactaran en los resultados de la búsqueda.

5.1.3. Region

En la Figura 6 se puede ver el diagrama de clases para el paquete *region*. Este paquete forma parte del componente "CombinatoryEngine" en la Figura 1 de la sección 3.

La principal responsabilidad de este paquete es brindar al sistema, la implementación de diferentes regiones (restricciones) combinatorias sobre secuencias de ARN. Para la primer versión de "vac-o" y según lo establece la especificación de requerimientos, se contemplan dos tipos de restricciones.

- Conservación de la estructura secundaria (*SSRegion*).
- Conservación del código genético (*GCRegion*).

El segundo caso, no merece mayor detalle a nivel de diseño. Por otro lado, para la restricción de conservar la estructura secundaria, vale la pena profundizar en que implica garantizar esta responsabilidad.

La clase *SSRegion* genera aquellas secuencias de ARN que conservan una estructura secundaria dada (*vaccine_structure*). Ya que la cantidad de secuencias que conservan una misma estructura secundaria podría ser eventualmente muy grande, se ofrecen dos restricciones complementarias para reducir el numero de secuencias que son tenidas en cuenta. A continuación resumimos brevemente cada una de estas restricciones:

- **Distancia mínima** a las secuencias que conservan una estructura secundaria dada (*wt_structure*). Aquellas secuencias que estén a menor distancia que la distancia mínima, serán descartadas. La cantidad de secuencias con las que se compara y calcula la distancia, lo determina el valor de *wt_seq_cache*.

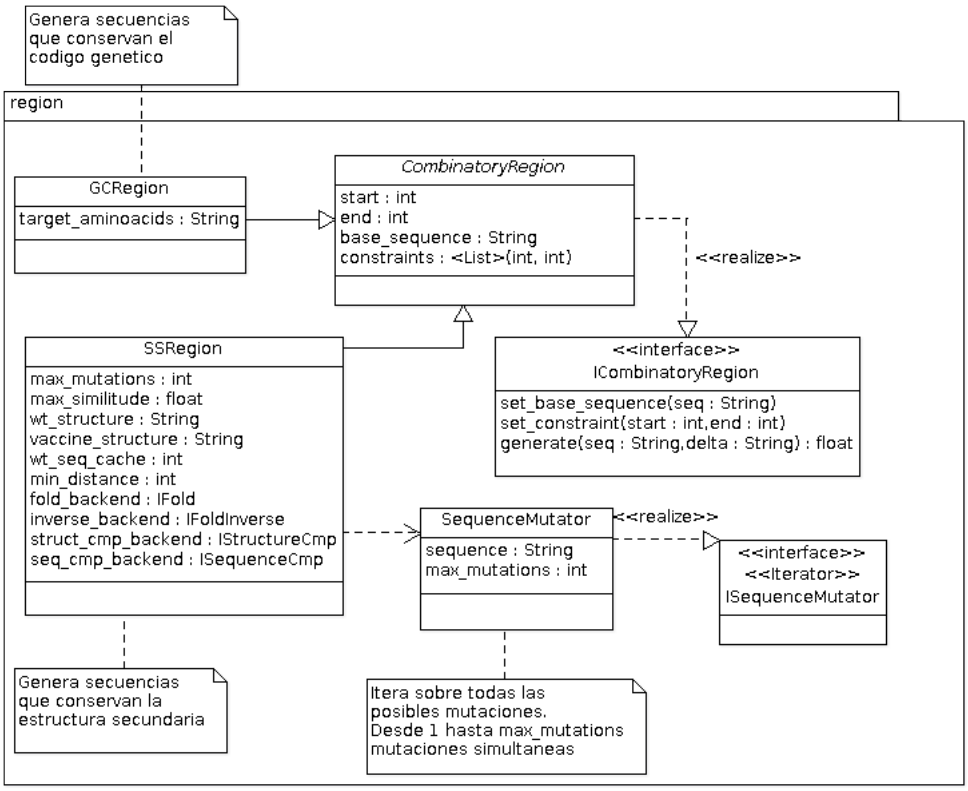

Figura 6: UML - Region

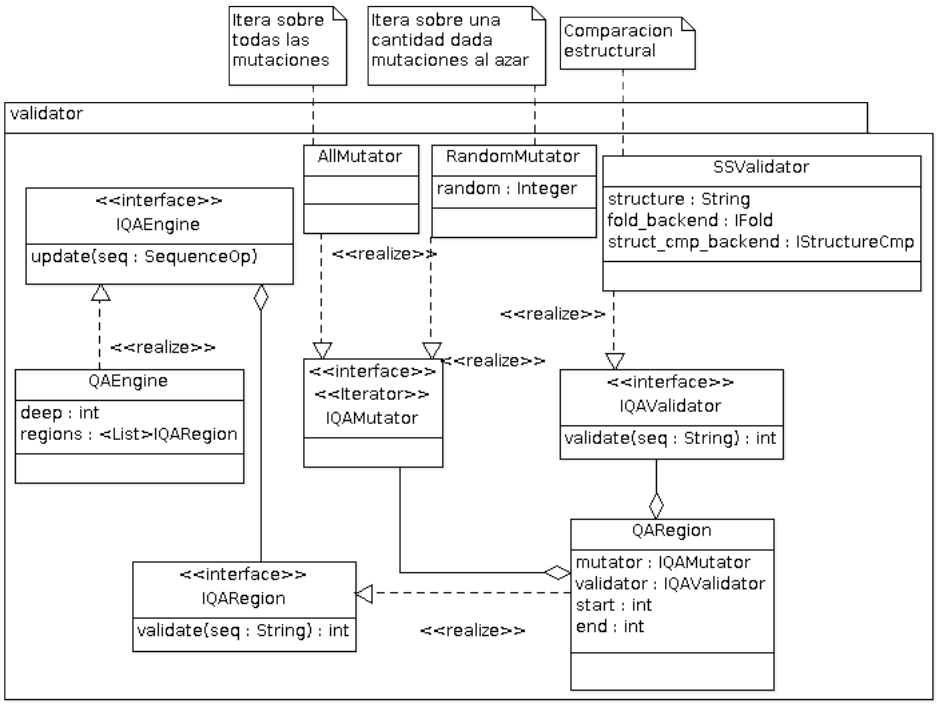

Figura 7: UML - Validator

- **Similitud estructural** a una estructura secundaria dada (*wt_structure*). Esta restricción implica un mayor costo computacional ya que partiendo de una posible secuencia, se realizan desde una, hasta *max_mutations* mutaciones simultaneas sobre toda la secuencia, y para cada posible mutación, se comprara la estructura secundaria de la mutación con la estructura *wt_structure*. Aquellas secuencias que excedan el porcentaje *max_similitude* de similitud, serán descartadas.

5.1.4. Validator

En la Figura 7 se puede ver el diagrama de clases para el paquete *validator*. Este paquete representa el componente "QAEngine" en la Figura 1 de la sección 3.

La responsabilidad de este paquete es realizar una serie de pruebas que garanticen al sistema que una secuencia dada, merece ser evaluada y tenida en cuenta como posible optimización de la vacuna. Las pruebas que se realizan para garantizar la "calidad", se basan en que luego de sufrir alguna cantidad acumulada de mutaciones, la secuencia mantenga determinadas propiedades en determinadas regiones.

En la primer versión de "vac-o" se contemplan dos maneras de generar mutaciones de una secuencia:

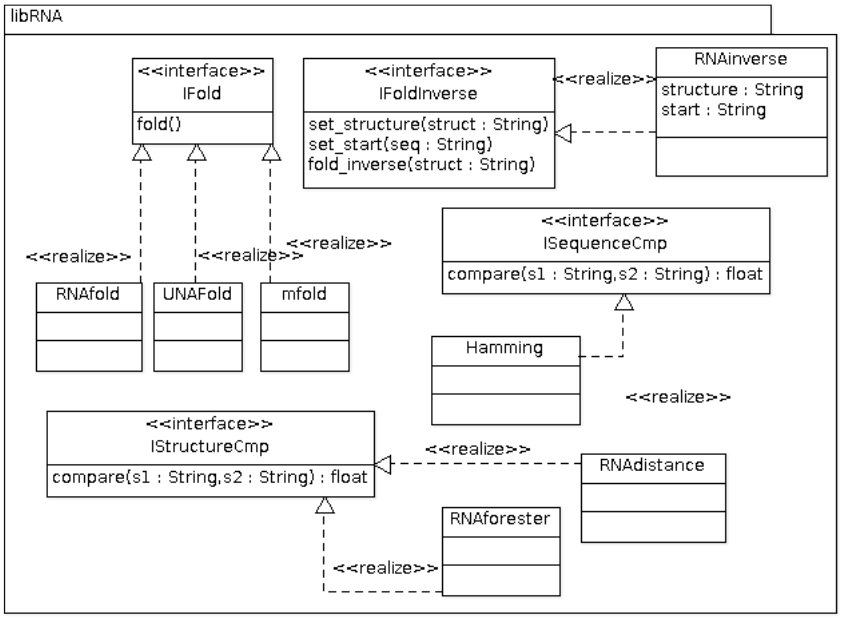

Figura 8: UML - LibRNA

- Todas las mutaciones posibles (*AllMutator*)
- Una cantidad (*random*) de mutaciones al azar (*RandomMutator*)

y dos propiedades a verificar sobre cada mutación:

- Similitud y Disimilitud estructural (*SSValidator*).

5.1.5. LibRNA

En la Figura 8 se puede ver el diagrama de clases para el paquete *libRNA*. Este paquete representa el componente "libRNA" en la Figura 1 de la sección 3.

La responsabilidad de este paquete es brindar al sistema servicios para la manipulación de secuencias de ARN. Para cumplir con esta responsabilidad, se ofrece al sistema el acceso a librerías externas de manera transparente y permitiendo utilizar diferentes librerías para acceder a diferentes servicios.

En la primer versión del sistema, se contemplan los siguientes servicios como indispensables, aunque en futuras versiones se podrían agregar otros:

- "Folding" directo (*IFold*)
- "Folding" inverso (*IFoldInverse*)
- Comparación entre estructuras (*IStructureCmp*)
- Comparación entre secuencias (*ISequenceCmp*)

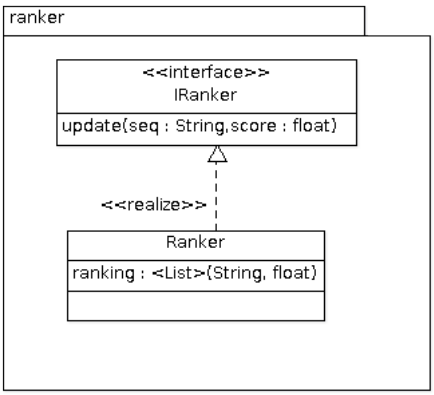

Figura 9: UML - Ranker

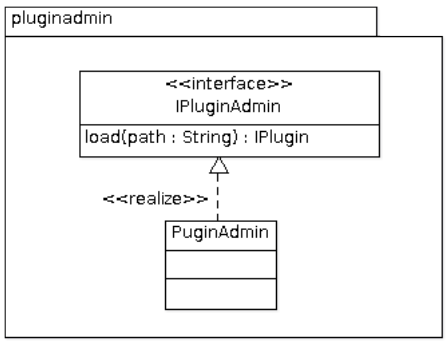

Figura 10: UML - PluginAdmin

La importancia de este paquete y las interfaces que contiene radica en que le permite al resto del sistema, abstraerse del uso de una u otra librería, y contar con una API unificada para acceder a estos servicios.

5.1.6. Ranker

En la Figura 9 se puede ver el diagrama de clases para el paquete *ranker*. Este paquete representa el componente "Ranker" en la Figura 1 de la sección 3.

La responsabilidad de este paquete es mantener el *ranking* de secuencias candidatas a optimizar la vacuna.

5.1.7. PluginAdmin

En la Figura 10 se puede ver el diagrama de clases para el paquete *pluginadmin*. Este paquete no aparece explícitamente en la arquitectura del sistema (Figura 1 de la sección 3), pero lo podemos identificar con la flecha que une el componente "Main" con "Plugin".

Fundamentalmente la responsabilidad de este paquete, es brindar al sistema la funcionalidad de cargar las extensiones en memoria.

5.1.8. Plugin

Para el paquete *plugin* no se especifica un diseño de bajo nivel debido a que la implementación de cada extensión no forma parte del sistema "vac-o". Simplemente, se asume que las extensiones que se implementen en el futuro, deberán garantizar que cumplen con el "contrato" establecido por las interfaces de este paquete.

Referencias

[1] Design Principles and Design Patterns. Robert C. Martin, 2000. `www.objectmentor.com`

[2] Rebecca Wirfs-Brock and Alan McKean. Object Design: Roles, Responsibilities and Collaborations, Addison-Wesley, 2003

[3] Unified Modeling Language: `http://www.uml.org/`

[4] ArgoUML: `http://argouml.tigris.org/`